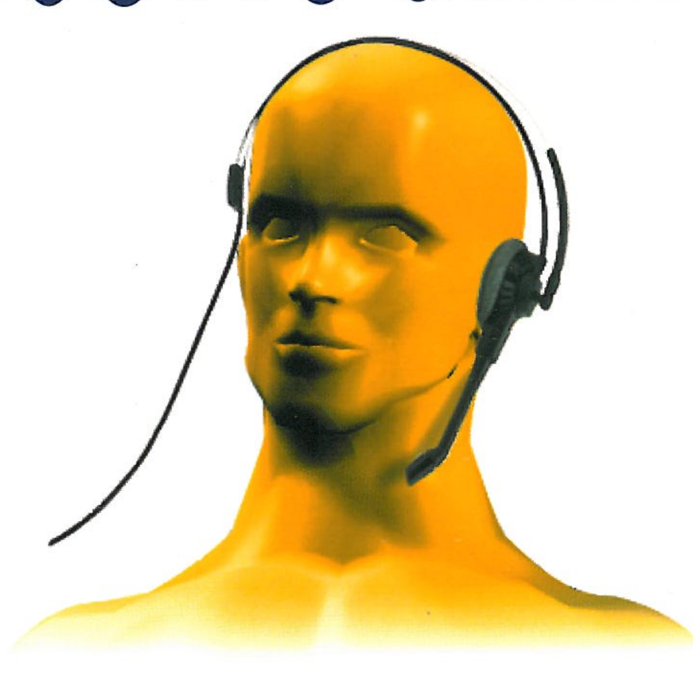
A PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

O V, NÚMERO 48

975 Ptas.

ESARROLLAR APLICACIONES CON RECONOCIMIENTO DE VOZ



C/C++
Programación básica en Windows (II)

VISUAL BASIC
Programación del API de Windows (I)

DIRECTX
Programación de DirectX con Delphi (y VIII): Joysticks analógicos y digitales

PROGRAMACIÓN DE SOCKETS
Programación de WinCGI (IV)

LINUX
Profundizando en SVGALIB (II)

JAVA
Utilización de threads (I)

C++ BUILDER 3
Creación de componentes

CONTENIDO DEL CD-ROM

- DirectX Media SDK v5.2
- SDKs de programación Internet
- PowerBuilder Code Analyzer v1.2b
- Catalyst 3 Application Generator
- Genitor v3.2 System Installation Kit

TOWER

8 413042 303299

SÓLO PROGRAMADORES

Número 48
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS
Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo de Riquer
eriquer@towercom.es
Redactora Jefe
Cristina Peña
crisp@towercom.es
Colaboradores
Ernesto Schmitz, Chema Alvarez, Constantino Sánchez Ballesteros, Enrique de la Lastra, Jordi Agost, Enrique de Alarcón, Jorge Delgado, Javier Sanz, Francisco M. Rosado.
Maquetación
Susana Llano
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29 - 970 47 44 35

Suscripciones
Begoña Alvarez
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
oscarrf@towercom.es
Javier Amado
jamado@towercom.es
Servicio Técnico
Óscar Casado
ocasado@towercom.es

Filmación
Megatipo
Impresión
Gráficas Reunidas
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN

EDITORIAL

El defecto 2000

Realmente dudo que el problema del fin del milenio (sin ningún tipo de doble intencionalidad), haga el número dos mil de la larga lista de fallos, errores o simplemente falta de previsión. Es posible que hablar de poca visión de futuro hace veinte años sea bastante injusto, sobre todo teniendo en cuenta las características de los PCs de dicha época, donde la velocidad, memoria y capacidad de disco ofrecían cifras, al igual que en la actualidad, acordes con la tecnología del momento. Algunos de estos fallos iniciales, como el caso de la FAT utilizada bajo MS-DOS para la gestión y almacenamiento de ficheros en disco no es un problema que aparezca ahora, sino que ya tuvo que resolverse a mitad de camino. Esta sensación de "lo soluciono y ya lo arreglaré de nuevo cuando dé problemas" no es producto de la dejadez, sino de la imposibilidad de predecir cómo evolucionará la tecnología actual a partir de la disponible hoy en día.

En el caso del problema de moda, los efectos ya se están padeciendo en algunas grandes empresas, cuyas reservas o pedidos se remontan a esas fatídicas fechas. Por esto, muchas emplean sus recursos en simular la situación para anticiparse a una serie de problemas graves debidos a la dificultad que supone encontrar fallos en un sistema que funciona aparentemente bien, pero que devuelve respuestas con información errónea.

Hasta tal punto llega la preocupación que se han incrementado de forma notable los seguros para cubrir las consecuencias de este problema y donde los informáticos responsables se enfrentarán a litigios por pérdidas originadas por su "torpeza". Sí, porque además, el hardware también se verá afectado, ya que el microprograma almacenado en sus chips también padecerá la misma problemática. Como consecuencia de esto, aparecerán beneficiados y damnificados, y aunque nuestra posición puede resultar incómoda en el sentido comentado anteriormente, también es cierto que la demanda de profesionales para solventar este efecto, en un mismo momento y en todo el mundo, supondrá una oportunidad para muchos programadores. Además, muchas empresas convierten en virtud este defecto y aprovechan para actualizar el software que se ejecuta en sus máquinas, que en muchos casos (más de los que podríamos pensar) trabajan con lenguajes similares al COBOL. Es decir, como las modificaciones y la necesidad de profesionales externos parece inevitable, actualicemos el sistema en función de los tiempos que corren.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: solop@towercom.es, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos. Sois nuestros mejores colaboradores.

SÓLO PROGRAMADORES



26

Programación de sockets

PROGRAMACIÓN DE WINCGI (IV)

Servidor y navegador son capaces de mantener un método de diálogo gracias a la interfaz CGI, la cual cuenta con una importante capacidad de posibilidades; y para que podamos demostrar la posibilidad de esa comunicación, el artículo nos acerca un ejemplo práctico.

46

Linux

PROFUNDIZANDO EN SVGALIB (II)

El entorno SVGALIB cuenta con muchas más posibilidades de las que pueda parecer a primera vista. Con esta librería el usuario puede acceder al sistema gráfico de su PC y dejar volar su imaginación. A lo largo del artículo descubriremos nuevas expectativas a la hora de crear.

63

C++ Builder 3

CREACIÓN DE COMPONENTES (I)

Con este artículo damos el pistoletazo de salida a una nueva serie destinada a la programación y desarrollo de componentes C++ Builder. A partir de este momento seremos capaces de crear aplicaciones basándonos en estos elementos.

72

Libros

ULTIMAS NOVEDADES

En esta ocasión os mostramos cuatro nuevos manuales que os pondrán al día sobre temas tan interesantes como ActiveX o Visual J++, además de adelantaros los posibles problemas que se suscitarán en el sector informático con la llegada del año 2000, o la inminente aparición de un entorno como Windows 98.

56 Java

UTILIZACIÓN DE THREADS (I)

La utilización del lenguaje Java nos permite desarrollar programas con threads (hilos) de una manera eficaz, con lo que se abre un nuevo espacio en el mundo de la pro gramación. Inauguramos una nueva serie en la que aprenderemos todo lo que hay que saber respecto al uso de éstos.

74

CORREO DEL LECTOR

En esta sección se pretende dar respuesta a todas las dudas que puedan tener nuestros lectores, pudiendo dirigirse a los propios autores o bien a la siguiente dirección de correo de nuestra publicación: solop@towercom.es

77

CONTENIDO DEL CD-ROM

Como es habitual nuestra revista os ofrece un nuevo CD-ROM con todo aquello que os interesa para completar la información de los artículos y lo más novedoso en el campo de las herramientas de programación. Catalyst 3, Genitor o PowerBuilder son sólo un adelanto de todo lo que podéis encontrar.



Se llama Irix 6.5

SILICON GRAPHICS LANZA LA 5ª GENERACIÓN DE UNIX DE 64 BITS

Ya está en la calle Irix 6.5, la última versión de su sistema operativo UNIX. Un producto preparado para el año 2000 y compatible binario con versiones anteriores.

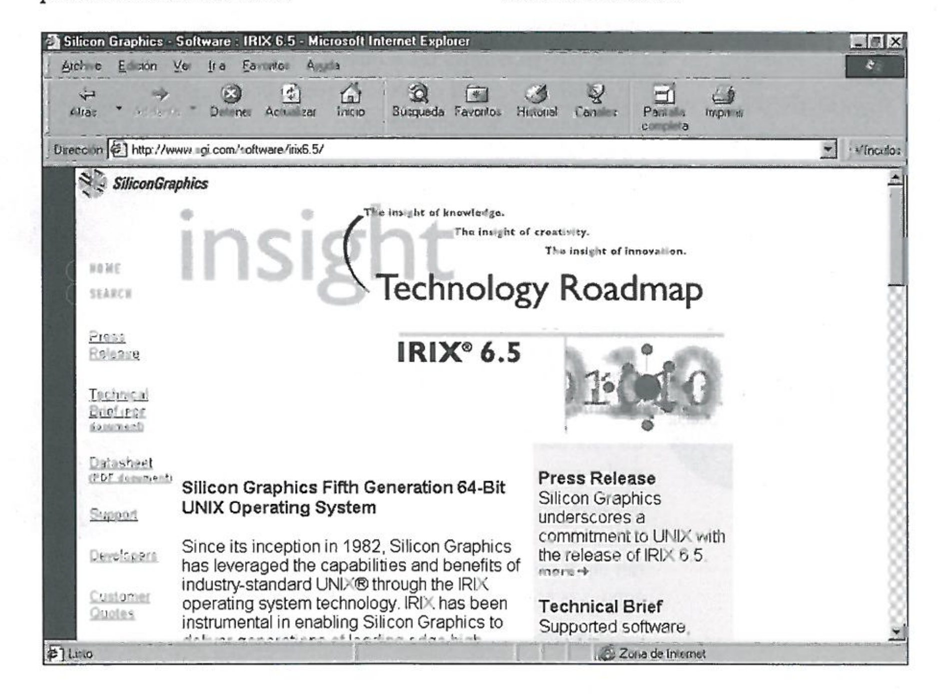
Incorpora características interesantes como la escalabilidad CCNUMA y funcionalidades mejoradas para manejo de grandes sistemas que reducen los costes totales de adquisición de un sistema de proceso de datos.

El nuevo sistema mejora la funcionalidad y el rendimiento de toda la línea de estaciones de trabajo, servidores y supercomputadores del propio Silicon Graphics, unificando y equilibrando la administración de todos sus sistemas.

Recientemente Silicon ha llegado a acuerdos con empresas líderes de informática de consumo como Computer Associates, Enlighten o Hewlett Packard, lo que junto con el rendimiento distribuido, la interoperabilidad y la gestión integrada de los sistemas Silicon Graphics, los harán soluciones muy interesante para entornos empresariales en general.

Incluye características como la escalabilidad de hasta 128 procesadores, una mayor interoperabilidad entre plataformas heterogéneas y una compatibilidad binaria total.

Además el nuevo Irix 6.5 incluye diversas herramientas que permiten al sistema el tratamiento de ficheros XFS de 64 bits, simplicidad para la gestión de los Centros de Proceso de Datos incluyendo herramientas como MISER para la planificación y asignación de recursos.



SUN ADQUIERE REDCAPE POLICY SOFTWARE PARA MEJORAR LAS FUNCIONALIDADES DE SU SOFTWARE DE ALMACENAMIENTO

Sun Microsystems ha anunciado la adquisición de la empresa Redcape Policy Software, lo que va a permitir el ampliar de forma significativa sus funcionalidades de software de gestión del almacenamiento corporativo.

Redcape desarrolla software de control automático para entornos abiertos con tecnología Java.

Sun piensa integrar el entorno de gestión y control de Redcape en su software de almacenamiento Sun StorEdge a fin de llegar a crear soluciones de software inteligentes que sean capaces de solventar automáticamente los posibles problemas de coste e ineficiencia de administración actual, para que exista la posibilidad de que simplifique la compleja gestión del almacenamiento con procesos de automatización.

Policy Management ofrece una tecnología emergente que juega un papel clave en la gestión de los recursos para el almacenamiento masivo y que permite a las empresas alcanzar los resultados u objetivos deseados mediante procedimientos automatizados previamente definidos por el usuario.

Con esta adquisición, Sun y sus aliados estratégicos disponen de una gran oportunidad para crear soluciones de gestión del almacenamiento inteligente e interoperables, con características que ofrecen a las empresas un alto valor. Para mayor información puede visitar la página web en **www.sun.es**

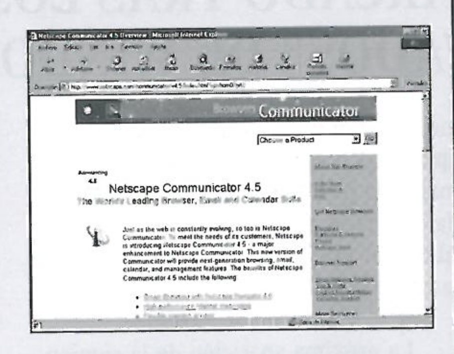
Entre malas noticias

NETSCAPE LANZA SU NUEVO COMMUNICATOR 4.5

En tan sólo seis meses, y según la información facilitada por la empresa AdKnowledge, la compañía Netscape cayó un 8.9% hasta llegar al 52.2% de la cuota mercado de los navegadores y en este mismo periodo Microsoft sumó un 9.6% hasta situarse en el 45.6%.

Entre tanta mala noticia Netscape anunció la salida de la nueva suite Communicator 4.5.

A partir de este momento contamos con la posibilidad de descargar la versión beta de la nueva herramienta en la página principal de Netscape.



Existen algunas direcciones de Internet a las que deben dirigirse en caso de querer solicitar mayor información, son las siguientes: **www.netscape.com/download/prev.html.**

Los productos de software AG la soportan TECNOLOGÍA DE COMPONENTES COM+ DE MICROSOFT

Dentro de su estrategia y compromiso con la tecnología de componentes, Software AG anuncia que sus productos soportarán la nueva generación de componentes COM+ de Microsoft. COM+ es una ampliación del estándar DCOM que ofrece una infraestructura técnica adecuada para que los desarrolladores puedan utilizarla como base para la creación de aplicaciones distribuidas, sin tener que preocuparse por los lenguajes de programación.

Entre las mejoras de COM+, se incluyen servicios, balance de carga, mejores funciones de seguridad, consultas en cola y servicio de base de datos en memoria que mejora el rendimiento cuando se accede a los componentes.

La tecnología de "componentware" es la base de los productos de desarrollo de Software AG. Con su producto EntireX, ha ampliado su tecnología de integración al incluir el estándar de componente DCOM. Asimismo Software AG y Microsoft han colaborado conjuntamente para conseguir que el estándar DCOM esté disponible sobre plataformas Unix y mainframes.

PERVASIVE Y SYNCHROLOGIC FORMAN EQUIPO PARA CREAR UNA SOLUCIÓN DE BASE DE DATOS SINCRONIZADA

Pervasive anuncia una unión estratégica con Synchrologic para integrar la tecnología de sincronización en el motor de base de datos ultra ligero Pervasive SQL.

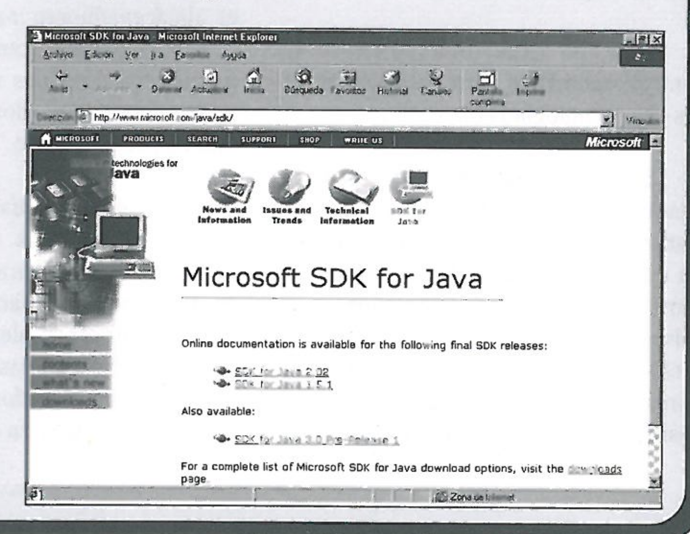
Este acuerdo les dará la posibilidad a los desarrolladores de sincronizar aplicaciones móviles y ocasionalmente conectadas, así como proporcionar de forma eficaz soluciones robustas de fácil implementación y prácticamente sin ninguna necesidad de administración.

La nueva tecnología permitirá a los desarrolladores diferentes posibilidades tales como sincronizar los datos móviles con sus aplicaciones basadas en Pervasive sin reescribir el código ni modificar el esquema de la base de datos, factores que aceleran el ciclo del desarrollo de productos, reducen las actividades de mantenimiento e incrementan la flexibilidad de las aplicaciones.

La combinación de la reconocida sincronización SyncKit de Synchrologic y del motor de base de datos ultra ligero de baja administración de Pervasive, permite a una amplia gama de aplicaciones ocasionalmente conectadas, sincronizar datos con un sitio central.

La tecnología permitirá también la sincronización de Pervasive SQL con datos de Oracle, Microsoft y Sybase.

Si usted está interesado en conocer más detalles de este acuerdo, las direcciones de Internet en las que puede ampliar la información respecto a este tema son: **www.pervasive.com** y **www.synchrologic.com.**



Mutismo de Microsoft
NT 5.0 HA SIDO APARCADO TRAS LOS "PRONÓSTICOS RESERVADOS" DE WINDOWS 98

El analizar las situaciones que acontecen tras los anuncios de nuevas versiones de los sistemas operativos Microsoft no es tarea fácil. Si además nos referimos al caso de NT comienza a ser aún peor.

Tras la puesta a la venta de Windows 98 y los primeros "pronósticos reservados" sobre la calidad que presenta este sistema, parece que han dado bastante en que pensar a la multinacional de Gates.

Las promesas basadas sobre la idea de una pronta aparición de NT 5.0 parece que se están desvaneciendo. En un principio parece ser que el código está casi finalizado por completo, pero la compañía no ofrece ninguna fecha exacta ni de salida del producto final ni tan siquiera de las últimas betas.

La próxima aparición de la siguiente versión para beta-testers estaba prevista con fecha inicial del mes de junio pero parece que tendremos que esperar hasta finales del verano para tenerla.

Y es que debemos contemplar que NT es una clave muy importante en el futuro más cercano de los sistemas operativos de Microsoft, ya que habrá que tener en cuenta que será el código que se utilizará de aquí en adelante tanto para las todas aquellas versiones profesionales como para los próximos sistemas de consumo.

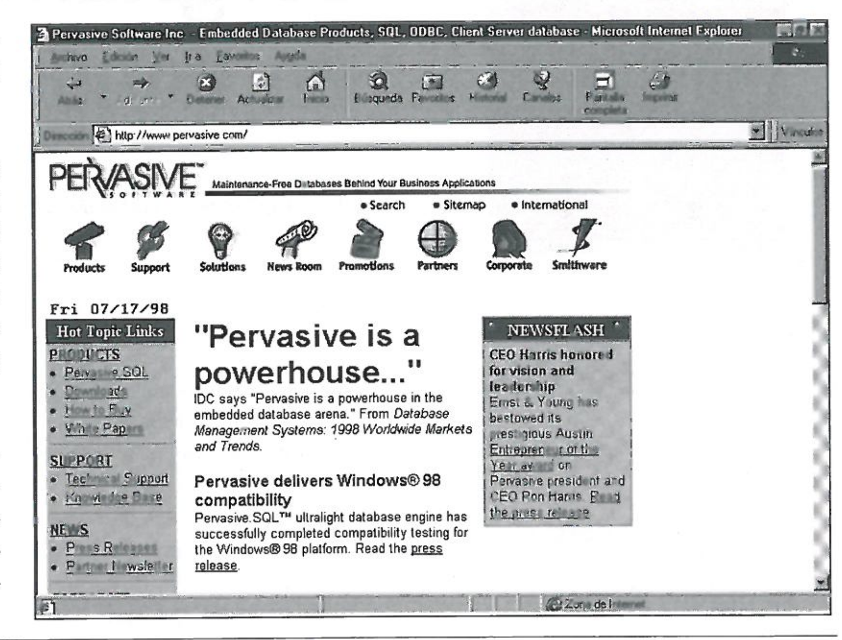
El mutismo que se ha creado en torno a este tema es alarmante; así que la pregunta que cualquier interesado tiene en mente no es otra que la siguiente: ¿Esperarán los expertos a ver los resultados de venta de Windows 98 para contarnos algo?

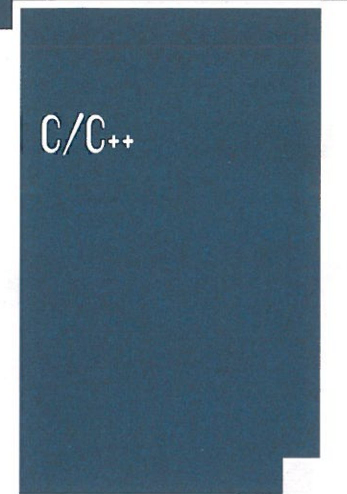
Nueva batalla por Java
SUN ALARMA A LOS DESARROLLADORES CONTRA MICROSOFT

Sun Microsystems ha comenzado una nueva batalla contra Microsoft y como vienen siendo habitual el punto de discrepancia se centra en Java.

Según las últimas informaciones de Sun, las nuevas implementaciones que ha hecho Microsoft de este lenguaje de programación incluye algunas importantes incompatibilidades. Además acusa a la multinacional de Bill Gates de intentar sabotear intencionadamente las premisas más importantes de este lenguaje, "se escribe una vez, se ejecuta en cualquier lugar".

Entre las acusaciones cabe destacar la denuncia de que Microsoft está reemplazando ciertas partes del lenguaje Java con un código específico de Windows para confundir a los desarrolladores y que éstos piensen que en realidad programan Java puro.





Programación básica en Windows (II)

Jorge Figueroa (erde@arrakis.es)

En el anterior número apareció el primer capítulo de este pequeño curso dedicado al desarrollo C bajo Windows.
En él, se explicaron muchos de los conceptos iniciales para adentrarse en este sistema operativo.
A partir de este momento, se van a especificar todas las líneas de código de un programa elemental.

Una vez explicados todos los conceptos generales de la programación Windows en el anterior artículo de esta pequeña serie, en el presente capítulo se van a explicar a fondo las líneas de código que aparecen en el programa de ejemplo que hay en el Listado 1, de forma que el lector pueda entender que hace cada línea de programa. Con ello, cada usuario podrá empezar a crear sus propios proyectos de práctica usando el programa de ejemplo como código base, pudiendo modificar de él todo lo que necesite para que se adapte a las características que desee.

REGISTRO DE CLASES DE WINDOWS

Cada ventana que se crea en un programa se basa en una *"clase de ventana"*.

Una *clase de ventana* consiste en un conjunto de estilos, fuentes, barras de títulos, etc... que el programador elige para usar en ella.

La clase *ventana* está definida como estándar en forma de estructura y la podemos encontrar en el fichero de cabecera *winuser.h*, donde la encontramos definida bajo el nombre de WNDCLASSW de la forma que sigue:

```
typedef struct tagWNDCLASSW
 {
  UINT style;
   WNDPROC  lpfnWndProc;
   int          cbClsExtra;
   int          cbWndExtra;
   HANDLE     hInstance;
   HICON       hIcon;
   HCURSOR  hIcon;
   HBRUSH     hbrBackground;
   LPCWSTR   lpszMenuName;
   LPCWSTR   lpszClassName;
 } WNDCLASSW, *PWNDCLASSW, NEAR
      *NPWNDCLASSW, FAR LPWNDCLASSW;
```

Windows posee varias clases de ventanas predefinidas (las más corrientes y susceptibles de poder ser usadas), aunque normalmente casi todos los programadores a la hora de crear sus aplicaciones suelen definir las suyas propias.

CbClsExtra indica los bytes extras que queremos asignar

A continuación se explican los diversos campos que contiene la estructura:

- cbClsExtra: Aquí se indica el número de bytes extras que queremos asignar a la propia estructura (para poder añadir más elementos). Por defecto contiene el valor NULL.

- cbWndExtra: Se usa para indicar la cantidad de espacio extra (indicado en bytes) que queremos reservar para las estructuras adicionales que queramos incluir en la propia clase de ventana, de forma que cuando se use esta clase se puedan añadir otros parámetros extras no definidos inicialmente. En situaciones normales vale NULL y no se usa más que en programación muy avanzada.

- hbrBackground: En este campo se define el pincel que se usará para pintar el fondo de la ventana. Por pincel tanto podemos entender un color sólido como una trama de textura. Los colores sólidos estándar de Windows que están permitidos son los que aparecen especificados en la Tabla 1.

- hCursor: Se usa para indicar el icono (puntero al icono) que se usará como gráfico del ratón cuando estemos dentro de la ventana (Normalmente vale NUL, o sea, el ratón por defecto de Windows).

- hIcon: Aquí podemos indicar qué icono se usará cuando la ventana esté minimizada (un puntero). Normalmente también está con el valor NULL.

La clase ventana está definida en el fichero winuser.h

- hInstance: Como ya se ha explicado, cuando arranca una aplicación, se obtiene un handle (de WinMain( )) que permite que quede identificada ante Windows, es el parámetro hInst. Este mismo handle, se debe dar a hInstance para que cuando se abra una ventana en la aplicación que use esa clase de ventana, Windows sea capaz de poder reconocer a que programa pertenece.

Tabla 1. Colores sólidos estándar de Windows.

COLOR_ACTIVEBORDER, COLOR_ACTIVECAPTION, COLOR_APPWORKSPACE, COLOR_BACKGROUND, COLOR_BTNFACE, COLOR_BTNHIGHLIGHT, COLOR_BTNSSHADOW, COLOR_BTNTEXT, COLOR_CAPTIONTEXT, COLOR_GRAYTEXT, COLOR_HIGHLIGHT, COLOR_HIGHLIGHTTEXT, COLOR_INACTIVEBORDER, COLOR_INACTIVECAPTION, COLOR_INACTIVECAPTIONTEXT, COLOR_MENU, COLOR_MENUTEXT, COLOR_SCROLLBAR, COLOR_WINDOW, COLOR_WINDOWFRAME, COLOR_WINDOWTEXT.

- lpfnWndProc: Gracias a este parámetro se puede enviar un parámetro a la función que se asocie a cada ventana creada que use esta clase para ser definida, el parámetro será de tipo puntero (de memoria). Esto pertenece a programación avanzada, por lo que de momento quedará sin detallarse más.

- lpszClassName: Es un puntero del tipo long que debe apuntar a una cadena con terminación nula. El contenido de la cadena corresponde al nombre de la clase de ventana y tendrá que ser único, de forma que así se eviten confusiones cuando una clase de ventana sea usada por más de una ventana.

- lpszMenuName: Aquí se indica un puntero tipo Long a la cadena que determina el nombre del recurso de un menú. La cadena debe acabar en NULL.

- Style: Especifica el estilo de la clase. Los atributos de estilo se pueden combinar a base de realizar OR's lógicos, dado que cada atributo de estilo corresponde a 1 bit diferente en la variable de Style.

LOS ATRIBUTOS DE STYLE

Los atributos que se pueden dar a Style son un poco complicados de explicar ya que muchos incluyen en si mismos gran cantidad de conceptos avanzados, por ello, y para evitar complicar las explicaciones, se dará sólo una breve explicación la función que realiza cada uno ellos:

- CSBYTEALIGNCLIENT: Este atributo hace utilizar límites de bytes en la dirección X. Alinea el área cliente de una ventana respecto al tamaño de la ventana que la contiene.

- CS_BYTEALIGNWINDOW: Utiliza un límite X en bytes en la dirección horizontal y alinea la ventana respecto al tamaño de la pantalla.

Todo programa tiene que realizar una llamada a WinMain( ) y a la función de reclamación

- CS_CLASSDC: Este atributo hace que se asigne a una clase de ventana su propio contexto de pantalla que podrá ser compartido por diferentes instancias.

- CS_DBCLKS: Hace que se envíe un mensaje de doble click a una ventana de forma automática.

- CS_GLOBALCLASS: Define una clase de ventana global.

- CS_HREDRAW: Si el tamaño horizontal (X) de la ventana se modifica-

se, se volverá a redibujar la ventana completa en pantalla.

- CS_KEYCVTWINDOW: Realiza una conversión virtual de teclas.
- CS_NOCLOSE: Hace que aparezca como desactivada (en gris) la opción de cerrar en el menú del sistema de la ventana.
- CS_NOKEYCVT: Desactiva una conversión de teclas virtual.
- CS_OWNDC: Asigna a cada una de las instancias de pantalla que haya su propio contexto de pantalla.
- CS_PARENTDC: Da al contexto de pantalla de la ventana origen a la clase ventana.
- CS_SAVEBITS: Almacena el área de imagen de una ventana que se encuentre tapada por otra para que pueda ser restaurada rápidamente de forma posterior.
- CS_VREDRAW: Si el tamaño Y de la ventana cambiase en un momento dado (porque el usuario ha modificado su tamaño moviendo los bordes por ejemplo), se redibuja toda (actualiza).

LAS DEFINICIONES REALIZADAS EN EL EJEMPLO DE PROGRAMA

En el programa de ejemplo aparecen muestras típicas de ejemplo de las definiciones detalladas anteriormente:

Si examinamos el código que hay en el ejemplo dado en la Listado 1, veremos que en él se asigna un nombre genérico al wcApp.lpszClassName de la ventana. Se debe asignar un nombre único de clase a cada clase nueva de ventana que se defina. Por otro lado, a hInstance se le asigna el valor devuelto en hInst, valor que devuelve el sistema después de que se haya ejecutado winmain( );.

Para crear una ventana antes tenemos que registrar su clase con RegisterClass( )

También tenemos lpfnWndProc, que como ya se ha explicado, es un puntero hacia la función de reclamación de ventana. En el EJEMPLO1, esta función es WndProc( );.

El campo wcApp.hCursor es un handle hacia el cursor, que para no complicarlo, se ha asociado con un cursor predeterminado de windows (IDC_ARROW). La asignación, como se puede observar en el código, se realiza con una llamada a la función LoadCursor().

Al campo wcapp.lpszMenuName se le da el valor 0, ya que el programa no posee menú alguno asociado.

Si la clase no queda registrada, se ejecuta return 0, con lo que finaliza el programa

Al campo wcApp.hbrBackground se le ha asignado un pincel, lo cual se realiza con una llamada a la función GetStckObject( ), la cual se encarga de retornar un handle (puntero) a un pincel predeterminado de Windows (LTGRAY_BRUSH);.

En el campo WcApp.style, se ha colocado los atributos CS_HREDRAW y CS_VREDRAW. Gracias a estos dos atributos, cada vez que el usuario cambie el tamaño X o Y de la ventana, ésta se redibujará automáticamente en pantalla.

Una vez están realizadas todas las definiciones, se tiene que registrar la clase, lo cual, como se puede apreciar en el código, se realiza mediante la función RegisterClass( ); , la cual devolverá un puntero que apuntará a la clase. Si la clase no queda registrada, la aplicación ejecuta return 0, lo que finalizará el programa.

CREANDO UNA VENTANA

Una vez tenemos una clase ventana definida y la hemos registrado con RegisterClass( ); , tenemos que ésta no aparece aún en pantalla, ya que para ello es necesario crearla.

Una vez creada la ventana, tenemos que ejecutar ShowWindow( ) para que se dibuje en pantalla

Al crear una ventana, tenemos que llamar a la función CreateWindow( ); , la cual, además, podemos utilizar para dar información más concreta sobre las características que deberá tener (además de los datos que proporciona la propia clase de ventana).

Un ejemplo fácil de entender es el del propio programa de ejemplo, en el cual, CreateWindow( ); recibe información del parámetro que describe características como: la clase de ventana, el título, el estilo, posición inicial en pantalla, el handle base, el handle de menú y el handle de instancia:

hWnd= CreateWindow(szProgName, "Ejemplo 1. Un programa de ejemplo", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, (HWND) NULL, (HMENU) NULL, (HANDLE) hInst, (LPSTR) NULL);

Los parámetros indicados entre paréntesis en el código mostrado son fáciles de entender: szProgName define la clase de ventana. La cadena que hay a continuación es el texto que usará la barra de título de la ventana.

SzProgName es un parámetro que define la clase de ventana

Después de la cadena, tenemos un definidor de estilo de ventana. En este caso se ha usado WS_OVERLAPPEDWINDOW, que es un estilo estándar de Windows, y que en pantalla aparecerá como una ventana normal superpuesta con una barra azul de título, un menú de sistema y los iconos de control de tamaño de ventana y cierre de la misma.

Todos los parámetros que se han indicado a continuación, son valores CW_USEDEFAULT o NULL. De estos parámetros, el cuarto y quinto corresponden a la posición en la que aparecerá la ventana, los parámetros sexto y séptimo indican el tamaño de la ventana tanto X como Y, y los tres últimos parámetros son Handles: el de la ventana origen, el asignado al menú de la ventana y uno para indicar la posición de parámetros extras (que al no haber, se pone como NULL).

El bucle se encarga de recibir los mensajes procedentes del propio Windows

Listado 1. Listado con un programa básico de ejemplo (EJEMPLO 1).

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
Char szProgName[]= "ProgName";

Int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPreIns, LPSTR LpszCmdLine, int
nCmdShow)
{
 HWND hWnd;
 MSG lpMsg;
 WNDCLASS wcApp;

 WcApp.lpszClassName  = szProgName;
 WcApp.hInstance      = hInst;
 WcApp.lpfnWndProc    = WndProc;
 WcApp.hCursor        = LoadCursor(NULL, IDC_ARROW);
 WcApp.hIcon          = 0;
 WcApp.lpszMenuName = 0;
 WcApp.hbrBackground = GetStockObject(LTGRAY_BRUSH);
 WcApp.style          = CS_HREDRAW | CS_VREDRAW;
 WcApp.cbClsExtra     = 0;
 WcApp.cbWndExtra     = 0;
 If(!RegisterClass (&wcApp))
   Return 0;

 hWnd= CreateWindow(szProgName, "Ejemplo 1. Un programa de ejemplo", WS_OVER-
LAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, (HWND) NULL,  (HMENU) NULL, (HANDLE) hInst, (LPSTR)
NULL);

 ShowWindow(hWnd, nCmDShow);
 UpdateWindow(hWnd);
 While (GetMessage(&lpMsg, NULL, 0, 0))
    {
      TranslateMessage(&lpMsg);
      DispatchMessage(&lpMsg);
    }
  return(lpMsg.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT messg, WPARAM wParam, LPA-
RAM lParam)
{
 HDC hdc;
 PAINTSTRUCT ps;

 Switch (messg)
  {
   case WM_PAINT:
    hdc= BeginPaint( hWnd,&ps);

   // Funciones gráficas por debajo de GDI

   //Funciones gráficas por encima del GDI
   ValidateRect(hWnd, NULL);
   EndPaint(hWnd,&ps);
   break;

   case WM_DESTROY:
    PostQuitMessage(0);
    break;

   default:
   return(DefWindowProc(hWnd, messg, wParam, lParam));
  }
}
```

Tabla 2. Listado con los principales mensajes de Windows.

```
#define WM_ACTIVATEAPP 0x001C
#define WM_CANCELMODE 0x001F
#define WM_WM_CHARTOITEM 0x002F
#define WM_CLEAR 0x0303
#define WM_CLOSE 0x0010
#define WM_COMPACTING 0x0041
#define WM_COMPAREITRM 0x0039
#define WM_COPY 0x0301
#define WM_CREATE 0x0001
#define WM_CTLCOLORBTN 0x0135
#define WM_CUT 0x0300
#define WM_DESTROY 0x0002
#define WM_DEVMODECHANGE 0x001B
#define WM_DROPFILES 0x0233
#define WM_ENDSESSION 0x0016
#define WM_ENTERIDLE 0x0121
#define WM_ERASEBKGND 0x0014
#define WM_FONTCHANGE 0x001D
#define WM_FULLSCREEN 0x003A
#define WM_GETDLGCODE 0x0087
#define WM_GETFONT 0x0031
#define WM_GETHOTKEY 0x0033
#define WM_GETEXTLENGHT 0x000E
#define WM_HOTKEYEVENT 0x0045
#define WM_LBUTTONDBLCLK 0x203
#define WM_LBUTTONUP 0x0202
#define WM_MBUTTONDBLCLK 0x0209
#define WM_MBUTTONDOWN 0x0207
#define WM_MBUTTONUP 0x0208
#define WM_MDIACTIVATE 0x0222
#define WM_MDICASCADE 0x227
#define WM_MDITILE 0x0226
#define WM_MENUCHAR 0x0120
#define WM_MENUSELECT 0x011F
#define WM_MOUSEFIRST 0x0200
#define WM_MOUSELAST 0x0209
#define WM_MOUSEMOVE 0x0200
#define WM_NCACTIVATE 0x0086
#define WM_NCCALCSIZE 0x0083
#define WM_NCCREATE 0x0081
#define WM_NCPAINT 0x0085
#define WM_NCRBUTTONDBLCLK 0x00A6
#define WM_NCRBUTTONDOWN 0x00A4
#define WM_NCRBUTTONUP 0x00A5
#define WM_NULL 0x0000
#define WM_OTHERWINDOWCREATED 0x0042
#define WM_OTHERWINDOWDESTROYED 0x0043
#define WM_PAINT 0x000F
#define WM_PARENTNOTIFY 0x0210
#define WM_PASTE 0x0302
#define WM_POWER 0x0048
#define WM_QUERYDRAGICON 0x0037
#define WM_QUERYENDSESSION 0x0011
#define WM_QUERYOPEN 0x0013
#define WM_QUIT 0x0012
#define WM_RBUTTONDBLCLK 0x0206
#define WM_RBUTTONDOWN 0x0204
#define WM_RBUTTONUP 0x0205
#define WM_SETFONT 0x0030
#define WM_SETHOTKEY 0x0032
#define WM_SHOWWINDOW 0x0018
#define WM_SYSCOLORCHANGE 0x0015
#define WM_TIMECHANGE 0x001E
#define WM_UNDO 0x0304
#define WM_WINDOWPOSCHANGED 0x0047
#define WM_WINDOWPOSCHANGING 0x0046
#define WM_WININICHANGE 0x001A
```

Una vez ejecutada la función CreateWindow( ); , ésta devuelve un valor que corresponde al handle de la ventana. En caso de que no se creara correctamente por cualquier causa, la función devolvería el valor NULL.

DIBUJANDO Y ACTUALIZANDO LA VENTANA

Para dibujar la ventana creada en pantalla, tenemos una función específica.

ShowWindow() se usa para indicar el modo en que se quiere dibujar la ventana

ShowWindow( ) y en la cual sólo se han de indicar dos parámetros, el handle de la ventana (el devuelto por CreateWindow) y nCmdShow, que se usa para indicar como se ha de dibujar la ventana. Hay varios tipos predefinidos para este segundo parámetro, como son por ejemplo: SW_SHOWNORMAL (ventana normal), SW_SHOWMINNOACTIVE (ventana minimizada), etc...

La función de reclamación puede también procesar mensajes procedentes de Windows

Una vez hemos dibujado la ventana, tenemos que el área cliente ha quedado borrada, por lo que tendremos que redi-

bujarla también mediante la función UpdateWindow( ) (que genera un mensaje WM_PAINT) y la cual sólo necesita como parámetro el propio handle de ventana.

EL BUCLE DE MENSAJE

Lo siguiente que aparece en el programa de ejemplo es el llamado bucle de mensajes, ya que como se explicó en el anterior capítulo, las aplicaciones reciben todos los eventos que ocurren en el hardware y en el propio sistema mediante mensajes, por lo que esta parte de código resulta esencial:

```
While (GetMessage(&lpMsg, NULL, 0, 0))
  {
    TranslateMessage(&lpMsg);
    DispatchMessage(&lpMsg);
  }
```

La función GetMessage( ) es la que obtiene el mensaje que haya pendiente en la cola de mensajes de la aplicación (recordar que cada aplicación tiene la suya propia).

GetMessage() es la que obtiene la información que haya pendiente en la cola de mensajes de la aplicación

Su funcionamiento es sencillo: copia el mensaje dentro de la estructura que se le indica mediante el primer parámetro (&lpMsg).

El segundo parámetro, que está con el valor NULL, hace que la función devuelva cualquier mensaje de la cola que pertenezca a cualquiera de las ventanas pertenecientes al programa.

Los parámetros tercero y cuarto, se usan para indicar a la función que no aplique ninguna clase de filtro de mensajes, los cuales se usan cuando se quiere que la aplicación sólo reciba mensajes de clases específicas.

Dentro del bucle de mensajes tenemos dos funciones más: TranslateMessage( ) y DispatchMessage( ).

La primera, TranslateMessage( ), se utiliza para trasladar mensajes de teclas virtuales a mensajes de caracteres. Esta función sólo se requiere cuando la entrada del carácter tiene que ser procesada desde el teclado. En otras palabras, incluyendo esta función en el bucle se activa la interfaz de teclado.

TranslateMessage() se encarga del translado de mensajes

La segunda función: DispatchMessage( ), se usa para enviar el mensaje actual al procedimiento de ventana al que corresponda. En el caso de ser una aplicación con múltiples ventanas abiertas, esta función se encarga de mandar cada mensaje al procedimiento de ventana que le corresponda de forma totalmente automática.

EL RESTO DEL CÓDIGO DE LA APLICACIÓN

Todo programa Windows tiene que poseer una llamada a una función WinMain( ) y una llamada a la función de reclamación. Y precisamente eso es WndProc( ), una función de reclamación.

La función de reclamación es la que se ejecutará cada vez que Windows ejecute una operación sobre la ventana de la aplicación y en todas las ventanas del programa que se generen usando la misma clase de ventana.

Para resumirlo de alguna forma, podemos decir que esta función es la rutina asociada a una clase de ventana dentro de nuestro programa, y de ahí que sea obligatorio que haya una función como mínimo en un programa, ya que si no, sería como crear un programa vacío de código.

Todo programa Windows debe poseer una llamada a la función WinMain()

Además de ejecutar código de programa, la función de reclamación de una ventana puede además la posibilidad de recibir mensajes.

Todos los mensajes de Windows empiezan con los tres caracteres WM_L, y en la Listado 2 hay ejemplos de algunos de los mensajes típicos más usados.

PRÓXIMO CAPÍTULO

En este capítulo ha quedado explicado casi todo el código del programa dado de ejemplo. Ahora, si el lector dispone de algún libro de referencia de programación C para Windows o cualquier manual donde haya funciones explicadas, podrá ya empezar a practicar con él, modificándolo, incluyendo nuevas funciones, trozos de código, etc...

Para finalizar, sólo queda decir que en el próximo capítulo, se acabarán de explicar algunos pequeños detalles que se han de saber a la hora de crear cualquier programa Windows y se pasará a explicar la forma de usar nuevos recursos como son por ejemplo las barras de desplazamiento.

VISUAL BASIC

Programación del API de Windows (I)

Jordi Agost (agost@eup.udl.es)

Empezamos una serie de artículos, en los que poco a poco vamos a introducirnos en el mundo de la API, hasta llegar a utilizarlas según nuestras necesidades. De esta forma dotaremos a nuestros programas de Visual Basic de funcionalidades imposibles de realizar de otro modo. En este primer artículo veremos las bases del desarrollo posterior.

En primer lugar debemos desmitificar la Interfaz del Programación de Aplicaciones o *API* (*Application Programming Interface*). Diremos que la *API* es simplemente un grupo de funciones que el programador puede utilizar para implementar sus funciones. Por ejemplo, si escribimos un programa en *Dbase*, entonces todas sus funciones pueden considerarse la *API* de *Dbase*. Cuando hablemos entonces de la *API* de Windows, diremos que ésta es un conjunto de funciones, que forman parte de Windows y que están disponibles para todas las aplicaciones Windows.

Dichas funciones o librerías se enlazarán en tiempo de ejecución, y no de compilación con el consecuente ahorro en el espacio que ocupará el programa. Además tienen la ventaja de que las bibliotecas se pueden actualizar independientemente de la aplicación y diversas aplicaciones podrán compartir una misma DLL.

LAS APIS DE WINDOWS

Si nos encontramos programando en Visual Basic 5.0, al ser éste un entorno de desarrollo de 32 bits, tan sólo deberemos preocuparnos de las *APIs* de 32 bits, dependiendo de si nuestro sistema va encaminado hacia Windows NT o Windows 95. Pero si aún estamos programando en Visual Basic 4.0, al tratarse de un entorno de desarrollo para 16 y 32 bits, deberemos tener en cuenta las *APIs* de 16 bits y establecer declaraciones diferentes para cada una de ellas(a través de las directivas #If...Then...#Else que establecen una compilación condicional).

La API es un grupo de funciones usadas para implementar otras

A partir de este momento vamos a diferenciar las diferentes *APIs* que nos podemos encontrar en los diferentes sistemas operativos Windows:

- API Win16: La original de 16 bits, es la *API* de Windows 3.x y Windows para grupos de trabajo 3.x. Es soportada por Windows 95 y Windows NT en modo de 16 bits. Sólo se puede acceder a esta *API* desde las ediciones de Visual Basic 16 bits.
- API Win32: La *API* de 32 bits completa. Originaria de Windows NT, es posible utilizarla desde cualquier edición de Visual Basic para 32 bits o Visual Basic para Aplicaciones (VBA).
- API Windows95: Subconjunto de Win32 soportado por Windows 95.

Se puede llamar a todas las funciones de Win32 con esta *API*, aunque muchas no han sido implementadas, en especial las relacionadas con características de Windows NT que no están incluidas en Windows 95, como cabía esperar.

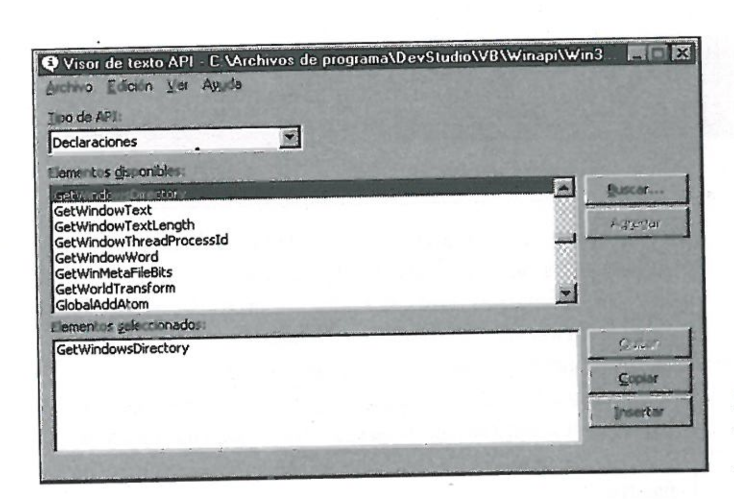
- API Win32s: Es un pequeño subconjunto de Win32 soportado por Windows 3.x y Windows para trabajo en grupo. No se soporta desde Visual Basic 4.0 y posteriores.

Las *APIs* de Windows poseen miles de funciones, procedimientos, tipos, que en cualquier momento podemos declarar y utilizar en nuestros proyectos. El lenguaje original de estos procedimientos es el lenguaje C/C++, por lo que deberemos declarar previamente cada procedimiento antes de poder utilizarlo con Visual Basic. Dichas declaraciones acostumbran a ser bastante complejas, razón por la cual la forma más sencilla es utilizar las declaraciones predefinidas que se incluyen en Visual Basic.

El Visor de API permite desplazarnos por las declaraciones del fichero

La forma para realizar lo explicado es la siguiente: dentro del subdirectorio \WinApi del directorio principal de Visual Basic, encontraremos el archivo Win32api.txt, el cual contiene declaraciones para muchos de los procedimientos de la *API* de Windows que se utiliza normalmente en Visual Basic. Para utilizar un determinado tipo, función, procedimiento u otra característica de este archivo, simplemente bastaría con copiar la línea que corresponda a un módulo de Visual Basic. Para ver dicho archivo podemos hacerlo desde cualquier procesador de textos o bien con el Visor de *API*, que es una utilidad incluida ya en Visual Basic.

La aplicación del Visor de API permite desplazarnos por las declaraciones,



La aplicación del visor de API.

constantes y tipos incluidos en el fichero de texto antes mencionado. Para visualizar la aplicación del Visor API, debemos ir al menú Complementos, luego al menú Administrador de complementos, y una vez allí asegurarnos que la casilla del Visor de API (*API Viewer*) se encuentra activada. Una vez hecho esto, la aplicación aparecerá en el menú *Complementos*.

Con Declare podremos obtener declaraciones

A continuación, ya dentro de la aplicación elegimos el menú Archivo y luego Cargar Archivo de Texto, eligiendo el archivo que deseemos cargar. Una vez esté cargado dicho archivo podremos visualizar las diferentes entradas en el cuadro de lista Elementos Disponibles, simplemente seleccionando Declaraciones, Constantes o Tipos en el cuadro de lista desplegable Tipo de API. Para buscar un elemento en concreto utilizaremos el botón Buscar. Además siempre podemos convertir el archivo de texto a un archivo de la base de datos Jet, para así optimizar la velocidad de la lista desplegable. Además pulsando una primera letra iremos hacía la declaración que deseemos. Y una vez hayamos encontrado la declaración deseada la copiamos en el portapapeles y la pegamos en nuestra aplicación.

Podríamos tener la tentación de incluir todo el archivo Win32api.txt en un módulo de Visual Basic, aunque es una práctica desaconsejable, ya que es un archivo grande y consumirá mucha memoria innecesaria de la aplicación, por lo que aconsejamos incluir tan solo las declaraciones que realmente vayan a ser utilizadas.

CREANDO NUESTRAS PROPIAS DECLARACIONES

A pesar de todo, tarde o temprano necesitaremos crear nuestra propias declaraciones, ya sea para acceder a librerías no estándar o bien para reescribir declaraciones predefinidas de Visual Basic con el fin de adaptarlas a nuestras necesidades. Para ello utilizaremos la instrucción Declare en la sección de declaraciones del código. Si dicha declaración pertenece a un módulo estándar entonces los procedimientos de DLL declarados serán públicos pudiéndose llamar desde cualquier parte de la aplicación. Si los declaramos en otro tipo de módulos entonces serán privados y deberemos declararlos como tales añadiendo al principio de la declaración la palabra clave Private.

Además bajo Win32 deberemos tener en cuenta que los nombres de las funciones contenidas dentro de las DLL o

Tabla. Principales DLLs y Biblioteca de extensiones para Windows.

Nombre de DLL	Función que desarrolla
ADVAPI32.DLL	Biblioteca de servicios avanzados de la API que admite múltiples API, incluidas llamadas de seguridad y del Registro.
COMCTL32.DLL	Incorpora algunos controles, tales como la lista en árbol y la edición de texto en formato RTF (Rich Text Format)
COMMDLG32.DLL	Donde se incluye el soporte para diálogos comunes.
GDI32.DLL	O Interfaz de Dispositivo Gráfico (Graphic Display Interface). Son funciones relacionadas con el dispositivo de salida, funciones de dibujo, de visualización del contexto, metaarchivos, coordenadas y fuentes.
KERNEL32.DLL	Posee funciones operativas de bajo nivel, para la administración de memoria, de tareas, y de recursos.
LZ32.DLL	Para compresión de archivos
MAPI32.DLL	Permite que cualquier aplicación pueda tener soporte para correo electrónico.
MPR.DLL	Biblioteca de enrutadores de múltiples proveedores.
NETAPI32.DLL	Para el acceso y control de redes de 32 bits.
ODBC32.DLL	Para la conectividad de bases de datos.
SHELL32.DLL	Biblioteca de la API para el shell en 32 bits.
USER32.DLL	Funciones relacionadas con la administración de Windows, tales como el manejo de menús, cursores, mensajes, control de tiempo, etc.
VERSION.DLL	Funciones relacionadas con el control de versiones.
WINMM.DLL	Ofrece acceso a las capacidades multimedia del sistema.

APIs son sensibles a las mayúsculas y minúsculas (al contrario que en Win16), por lo que deberemos tener cuidado si no queremos provocar errores en la ejecución del programa.

Podemos convertir el archivo de texto a un archivo Jet para optimizar la velocidad

Si el procedimiento que declaramos devuelve algún valor lo declararemos como si fuera una función (Function):

```
Declare Function nombre_público Lib
    "nombre_librería" [Alias "alias"] [([[ByVal]
    variable [As tipo] [,[ByVal] variable [As
    tipo]]—])] As Type
```

Si el procedimiento no devuelve ningún valor deberemos escribir la declaración como Sub:

```
Declare Sub nombre_público Lib "nombre_librería"
    [Alias "alias"] [([[ByVal] variable [As tipo]
    [,[ByVal] variable [As tipo]]—])]
```

donde :

nombre_público: el nombre del procedimiento de la DLL.

nombre_librería: la librería donde se encuentra dicho procedimiento. Si estamos haciendo referencia a alguna de las librerías comunes de Windows (User32, Kernel32 o GDI32) no necesitamos incluir la extensión del archivo, por ejemplo bastaría con la siguiente declaración:

```
Declare Function CloseClipboard Lib "user32"
    Alias "CloseClipboard" () As Long
```

Aunque también podemos especificar una determinada ruta:

```
Declare Function LZCopy Lib "lz32.dll" Alias
    "LZCopy" (ByVal hfSource As Long, ByVal
    hfDest As Long) As Long
```

Si no especificamos dicha ruta Visual Basic, buscará el archivo en los siguientes lugares:

1.- Directorio donde se encuentra la aplicación.
2.- Directorio actual.
3.- Directorio del sistema de Windows (usualmente \Windows\System).
4.- Directorio de Windows.
5.- Variable de entorno de la ruta de acceso.

En la siguiente tabla podemos observar las principales DLLs de las que está compuesto el sistema operativo Windows (32 bits), y las principales funcionalidades de sus procedimientos.

Si no especificamos la ruta VB buscará en diferentes directorios

nombre_alias: nos permite identificar la función con un nombre diferente del que tiene por sí misma, en nuestro programa de Visual Basic. Puede servir para tener acceso a funciones cuyo nombre no podía aceptar la sintaxis de VB, como por ejemplo la siguiente declaración:

```
Declare Function lopen Lib "kernel32" Alias
    "_lopen" (ByVal lpPathName As String,
    ByVal iReadWrite As Long) As Long
```

En dicho ejemplo lopen será el nombre con el que se conocerá a la función dentro del código de Visual Basic, y _lopen será el nombre del procedimiento dentro de la DLL.

También pude utilizarse si el nombre del procedimiento que invocamos

coincide con el de alguna palabra reservada, como en la siguiente declaración:

```
Declare Function SetFocused Lib "user32" Alias
        "SetFocus" (ByVal hwnd As Long) As Long
```

PROCEDIMIENTOS QUE TRATAN CON CADENAS

Con la API de Win32 el tratamiento de cadenas es ligeramente diferente, ya que una de las diferencias entre Win16 y Win32 es que Win32 admite hasta tres tipos de juegos de caracteres. Los dos primeros son de byte simple y byte doble. En el juego de caracteres de byte simple cada carácter está representado por un byte (con un máximo de 256 caracteres), y en el juego de byte doble (DBCS), algunos valores de los bytes están reservados, e indican que el byte que viene a continuación debe combinarse con otro, con lo que podemos romper la barrera de los 256 caracteres. El tercer tipo de caracteres es el Unicode, donde cada carácter se representa con 16 bits (dando un total de 65535 caracteres suficientes para todos los caracteres de todas las lenguas que están hoy en uso). Cada función de la API Win32 que utilice texto deberá determinar si el texto se encuentra en formato DBCS o Unicode.

Cada carácter Unicode se representa con 16 bits

La solución empleada por Microsoft a tal efecto es proporcionar dos funciones separadas para cada función de la API que utilice texto. Por ejemplo GetWindowText tiene en realidad dos formas distintas: GetWindowTextA y GetWindowTextW, donde A significa ANSI y W Wide (amplio o Unicode).

Windows NT utilitza Unicode internamente, por lo que si llamamos a la función GetWindowsTextA, en realidad Windows llamará a GetWindowsTextW y el valor retornado será convertido a caracteres ANSI. Por su parte Windows 95 no admite Unicode en ninguna forma, por lo que si llamamos a la función GetWindowsTextW se producirá un error. Visual Basic, aunque parezca sorprendente, siempre llamará a la versión ANSI de la función, a pesar de que internamente utiliza Unicode.

Una DLL se documenta siguiendo la sintaxis del lenguaje C++

Para evitarnos problemas relacionados con qué función llamar en cada caso, solamente debemos utilizar el nombre estándar de la API de Windows, dejando que sea Visual Basic, el que se las arregle para llamar en cada caso al nombre de la API que corresponda.

PASO DE ARGUMENTOS

Normalmente los procedimientos de una DLL se documentan siguiendo la sintaxis del lenguaje C++. Para lograr llamar a dichos procedimientos desde Visual Basic, debemos convertirlos en primer lugar en instrucciones Declare válidas y llamarlos con los argumentos correctos.

Un inconveniente será que Visual Basic no sabrá en ningún momento si estamos pasando los argumentos correctos

Tabla. Tipos de datos en C/C++ y su equivalencia en Visual Basic.

Tipos de datos en C/C++	Declaración en VB
ATOM	ByVal var As Integer
BOOL	ByVal var As Long
BYTE	ByVal var As Byte
CHAR	ByVal var As Byte
COLORREF	ByVal var As Long
DWORD	ByVal var As Long
HWND, HDC, HMENU	ByVal var As Long
INT,UINT	ByVal var As Long
LONG	ByVal var As Long
LPARAM	ByVal var As Long
LPDWORD	var As Long
LPINT, LPUINT	var As Long
LPRECT	var As tipo
LPSTR,LPCSTR	ByVal var As String
LPVOID	Var As Any
LPWORD	Var As Integer
LRESULT	ByVal var As Long
NULL	As Any o ByVal var As Long
SHORT	ByVal var As Integer
VOID	Sub procedimiento
WORD	ByVal var As Integer
WPARAM	ByVal var As Long

a un determinado procedimiento de una DLL. Si pasamos valores incorrectos, es más que probable que nuestro programa falle, con lo que deberemos reiniciarlo. Por lo tanto es muy aconsejable guardar frecuentemente nuestro trabajo cuando estemos utilizando llamadas a la API.

Visual Basic pasará todos los argumentos por referencia

En la siguiente tabla observamos los tipos de datos más comunes de C y su equivalente en Visual Basic (para Win32).

PASO POR REFERENCIA O POR VALOR

También deberemos tener en cuenta que, de forma predeterminada, Visual Basic pasará todos los argumentos por referencia. Es decir, en vez de pasar el valor real del argumento, Visual Basic pasa una dirección de 32 bits (puntero) en la cual se encuentra almacenado el valor del argumento. Podemos incluir la palabra clave ByRef en nuestras declaraciones si así lo deseamos, aunque no es necesario. En cambio nos encontraremos con que muchos procedimientos de DLL esperarán que les pasemos los argumentos por valor. Es decir, esperan el valor real, en lugar de su ubicación en la memoria. Si lo pasamos por referencia no funcionará. Para pasar un argumento por valor, lo precedemos de la palabra clave ByVal.

Las APIs de Windows no suelen utilizar el tipo de cadena BSTR

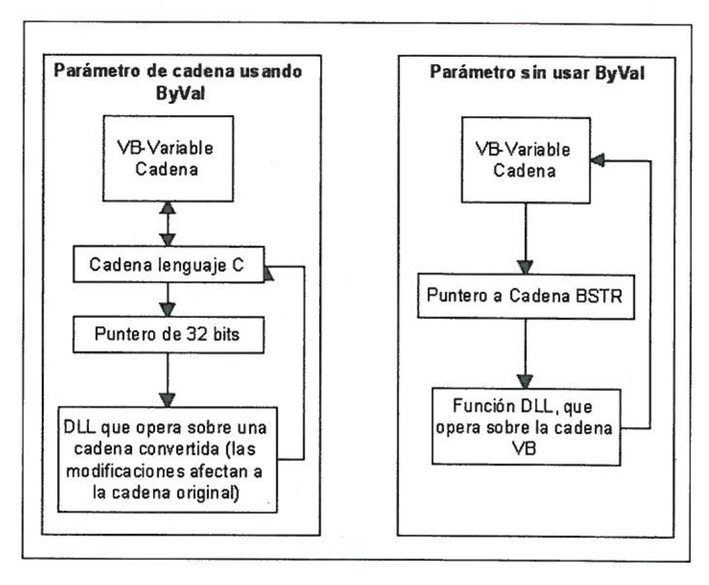
PASANDO CADENAS COMO ARGUMENTOS

Visual Basic es un poco inconsistente en el uso de ByVal con variables de tipo cadena. Existen dos formas de cadena, las cuales son admitidas para comunicarse entre Visual Basic y un procedimiento de una DLL. Si no aparece la palabra clave ByVal entonces Visual Basic empleará un tipo de datos String conocido como BSTR. Un tipo de datos BSTR (la mayoría de tipos de datos BSTR son Unicode) está formado por un encabezado que contiene información sobre la longitud de la cadena, y por la cadena propiamente dicha, la cual puede contener valores nulos. Este tipo de datos se pasará como un puntero (un puntero es una variable que contiene la ubicación en memoria de otra variable en lugar de los datos reales), lo que permitirá al procedimiento de la DLL modificar la cadena. Normalmente las APIS de Windows no utilizan el tipo de cadena BSTR, salvo aquellas que pertenecen a la API OLE 2.0, sino que las funciones API de Windows esperan que los parámetros de cadena sean pasados como un tipo LPSTR (Long Pointer String), es decir un apuntador a una cadena terminada con un nulo (cadena a veces llamada ASCIIZ), que es el tipo que se utiliza en C/C++ (Ver imagen).

Si un procedimiento de una DLL espera un LPSTR como argumento habremos de pasar del tipo de datos BSTR por valor (esto es, utilizando la palabra clave ByVal). Como los punteros del tipo BSTR son punteros al primer byte de datos de la cadena terminada con un carácter nulo, parecerá un tipo LPSTR para el procedimiento de la DLL.

Además, deberemos hacer un par de precisiones que generalmente conducen a un error: en primer lugar deberemos tener en cuenta que si pasamos datos binarios a un procedimiento de una DLL, habremos de pasar una variable como matriz de tipo de datos Byte, y no una variable de tipo String como pudiera pensarse, ya que se supone que las cadenas contienen caracteres y los datos binarios no se podrían leer correctamente en procedimientos de una DLL si los pasamos como variables String. Y en segundo lugar deberemos pensar que si declaramos una variable de cadena sin inicializarla previamente y la pasamos por valor a un procedimiento de una DLL, dicha variable se pasará como NULL, y no como pudiera parecer como una cadena vacía ("").

Para evitar este tipo de confusiones, es aconsejable que, siempre que queramos pasar un valor NULL a un argumentos de tipos LPSTR, utilicemos una constante que ya lleva el Visual Basic llamada vbNullString.



Tipos de datos BSTR y LPSTR para cadenas de tipo string.

MODIFICANDO CADENAS PASADAS COMO ARGUMENTO

Un procedimiento de una DLL determinada puede llegar a modificar los datos de una variable de tipo String o cadena que le haya sido pasada como argumento. Si dicha modificación sobrepasa la longitud que ya tenía la cadena, entonces el final de la cadena será sobreescrito y con toda seguridad otros datos adyacentes a ella serán dañados.

Muchos procedimientos de DLL no esperan como argumentos cadenas de más de 255 caracteres

La solución para evitar este problema consiste en pasar como argumento una cadena que tenga la longitud suficiente para que el procedimiento no la sobreescriba.

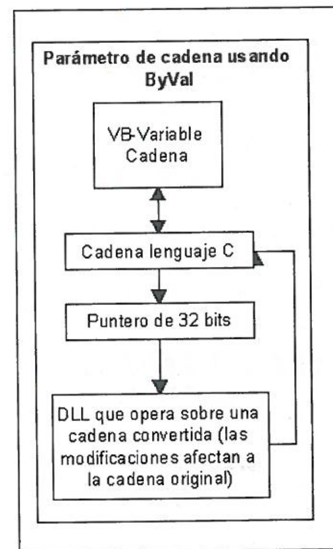
Por ejemplo para llamar a la siguiente función:

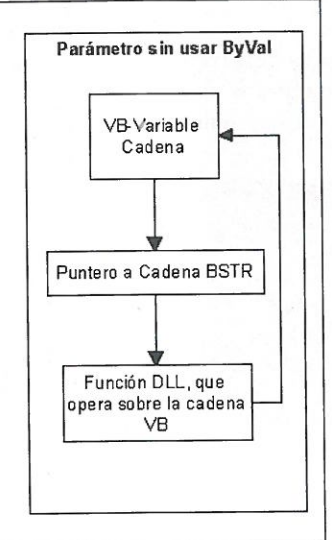
```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA"
        (ByVal lpBuffer As String, ByVal nSize As
        Long) As Long
```

Esta función retornará el directorio activo de Windows, lo podemos hacer de la siguiente manera:

```
Dim Path As String * 255
Ret = GetWindowsDirectory(Path, Len(Path))
```

Un punto a tener en cuenta es que muchos procedimientos de DLL no esperan como argumentos cadenas de más de 255 caracteres, aunque este dato siempre dependerá del procedimiento en cuestión.





Convenciones de llamada para parámetros de tipo String.

PASO DE PROPIEDADES

Las propiedades siempre deben pasarse por valor, si en la declaración un argumento ha estado declarado con la palabra clave ByVal. Por ejemplo, usamos la siguiente función para determinar en pixels la resolución de la pantalla o la impresora en un determinado momento:

```
Declare Function GetDeviceCaps Lib "gdi32" Alias
        "GetDeviceCaps" (ByVal hdc As Long,
        ByVal nIndex As Long) As Long
```

Declarándola en la sección de declaraciones de un módulo. Y en la sección de declaraciones del formulario añadimos las siguientes sentencias:

```
Option Explicit
Const PLANES = 14, BITS = 12
```

Las APIs de Windows poseen miles de funciones

En el formulario añadimos un botón. Y en el código correspondiente al evento clic de un botón (en este caso en particular llamado cmdCommand1), añadimos el siguiente código:

```
Private Sub cmdCommand1_Click()
MsgBox "Colores de la impresora: " & _
GetDeviceCaps(Printer.hdc, PLANES) * 2 ^
        GetDeviceCaps(Printer.hdc, BITS)
End Sub
```

Si lo que queremos es pasar una propiedad pero por referencia (en el argumento aparecerá la palabra clave ByRef) deberemos utilizar siempre una variable intermedia. Recordemos el anterior ejemplo, en el cual recuperábamos el directorio activo de Windows. Si lo que quisiéramos fuera asignar directamente la propiedad Path de un control de cuadros de una lista de archivos, podríamos hacer el siguiente código:

```
Dim Path As String * 255
Ret = GetWindowsDirectory(File1.Path,
        Len(File1.Path))
```

Aunque tal y como comentábamos, debido a la ausencia de la variable intermedia, dicho código no funcionaria. Deberíamos reescribirlo de la siguiente forma:

```
Dim Tmp$, Ret%
Tmp = String(255,0)
Ret = GetWindowsDirectory (Tmp, Len(Tmp))
Tmp = Left(Tmp, Ret)
File1.Path = Tmp
```

Hasta aquí hemos visto los principios básicos sobre el mundo de la API, aunque en posteriores artículos iremos adentrándonos cada vez más en este tema para dotar a nuestros programas de una mayor funcionalidad.

```
  Button2.Enabled    := False;
  Trackbar1.Enabled  := True;
  Trackbar2.Enabled  := True;
end;
```

Los procedimientos que iremos viendo a continuación se encargan de comprobar los valores de joysticks digitales. El que viene a continuación chequea si el joystick se encuentra centrado, es decir, que no tiene movimiento en ninguna de las direcciones por parte del usuario. En caso afirmativo, activamos en color azul el panel central para tener una referencia visual:

```
procedure
    TForm1.DGCJoystick1Joy1Center(Sender:
    TObject);
begin
  Panel1.Color := clNavy;
  Panel2.Color := clNavy;
  Panel3.Color := clNavy;
  Panel4.Color := clNavy;
  Panel5.Color := clBlue; //PANEL CENTRAL
  Panel6.Color := clNavy;
  Panel7.Color := clNavy;
  Panel8.Color := clNavy;
  Panel9.Color := clNavy;
end;
```

Si la palanca del joystick se ha movido hacia abajo, activamos su panel correspondiente:

```
procedure
    TForm1.DGCJoystick1Joy1Down(Sender:
    TObject);
begin
  Panel1.Color := clNavy;
  Panel2.Color := clNavy;
  Panel3.Color := clNavy;
  Panel4.Color := clNavy;
  Panel5.Color := clNavy;
  Panel6.Color := clNavy;
  Panel7.Color := clNavy;
  Panel8.Color := clBlue; //PANEL DE ABAJO
  Panel9.Color := clNavy;
end;
```

Existirá un procedimiento análogo a los anteriores asociado a cada uno de los movimientos de la palanca, tal y como aparece al principio del programa. Con el fin de no repetir estos procedimientos, únicamente indicaremos qué panel debe activarse en cada caso:

- Movimiento hacia abajo e izquierda:

```
Panel7.Color := clBlue; //PANEL ABAJO-
    IZQUIERDA
```

- Movimiento hacia abajo y derecha:

```
Panel9.Color := clBlue; //PANEL ABAJO-DERE-
    CHA
```

- Movimiento hacia la izquierda:

```
Panel4.Color := clBlue; //PANEL IZQUIERDO
```

- Movimiento hacia la derecha:

```
Panel6.Color := clBlue; //PANEL DERECHO
```

- Movimiento hacia arriba:

```
Panel2.Color := clBlue; //PANEL ARRIBA
```

- Movimiento hacia arriba e izquierda:

```
Panel1.Color := clBlue; //PANEL ARRIBA-
    IZQUIERDA
```

- Movimiento hacia arriba y derecha:

```
Panel3.Color := clBlue; //PANEL ARRIBA-DERE-
    CHA
```

Una vez terminados los procedimientos de representación visual de los joysticks digitales, pasaremos a la gestión de los modos de comprobación (analógico y digital). La selección de cada modo se efectúa mediante dos Radiobuttoms representados en el formulario.

Al activar el formulario debemos establecer unos valores por defecto

El siguiente procedimiento activa el modo digital y esconde del formulario el panel que representa el modo analógico (panel14). Mediante el método Joy1Mode asignamos dicho modo:

```
procedure TForm1.RadioButton1Click(Sender:
    TObject);
begin
  RadioButton2.Checked := False;
  RadioButton1.Checked := True;
  Panel14.Hide;
  DGCJoystick1.Joy1Mode := Digital;
end;
```

Si se selecciona el modo analógico, se llamará automáticamente a este procedimiento. En este caso, el valor del método Joy1Mode será Analog:

```
procedure TForm1.RadioButton2Click(Sender:
    TObject);
begin
  RadioButton1.Checked := False;
  RadioButton2.Checked := True;
  Panel14.Show;
  Panel14.BringToFront;
  Label5.Left := 40;
  Label5.Top  := 37;
  DGCJoystick1.Joy1Mode := Analog;
end;
```

El siguiente procedimiento activa o desactiva la propiedad Notify mediante su casilla de verificación incrustada en el formulario. Si el valor del método Joy1Notify es True, se nos notifica cuando ocurra un cambio en el estado del joystick. Por el contrario, si el valor de este método es False, se utilizarán intervalos de frecuencia del Polling:

```
procedure TForm1.CheckBox1Click(Sender:
    TObject);
begin
 If CheckBox1.Checked then
   DGCJoystick1.Joy1Notify := True
 else
   DGCJoystick1.Joy1Notify := False;
end;
```

Cuando se activa el formulario por primera vez, debemos establecer unos valores por defecto. Lo primero es asignar las frecuencias máximas y mínimas de polling. Esta información la obtenemos de las características del joystick mediante el método Joy1Caps. Aplicando a éste los métodos wPeriodMin y wPeriodMax podemos asignar los valores de polling para el dispositivo utilizando los valores de la barra de desplazamiento del formulario (valores con un rango de 0 a 1000).

La selección de cada modo se efectúa mediante dos Radiobuttoms que aparecen en el formulario

El siguiente paso es ajustar los valores mínimo y máximo de la barra de desplazamiento que representa al Threshold (distancia a la que el joystick debe moverse antes de que se envíe una notificación). Hay que tener en cuenta que el Threshold será ignorado en el modo digital.

Los nueve paneles azules representan las nueve posiciones del joystick digital

Por último, establecemos el rango que utilizaremos para situar el signo + en el panel correspondiente (modo analógico). Este panel es de aproximadamente 80*80 pixels. Para que el signo + se mueva por su panel, necesitamos encontrar la distancia a la que el joystick debe moverse por cada píxel. Por ejemplo, si el valor máximo es 65535, el mínimo 0, y nuestro panel 80, el cálculo debería rondar el valor de 820. Utilizaremos este valor cuando calculemos en otro procedimiento la posición del joystick relativa al panel:

```
procedure TForm1.FormActivate(Sender:
      TObject);
begin
  TrackBar2.Min :=
      DGCJoystick1.Joy1Caps.wPeriodMin;
  TrackBar2.Max :=
      DGCJoystick1.Joy1Caps.wPeriodMax;
  TrackBar2.Position :=
      DGCJoystick1.Joy1Polling;
  Label3.Caption := IntToStr(Trackbar2.Position);

  Trackbar1.Min := 0;
  Trackbar1.Max := 1000;
  Trackbar1.Position :=
      DGCJoystick1.Joy1Thresh;
  Label4.Caption := IntToStr(Trackbar1.Position);

  //Rangos para el eje X e Y
  XRange := (DGCJoystick1.Joy1Caps.wXMax -
      DGCJoystick1.Joy1Caps.wXMin) div 80;
  YRange := (DGCJoystick1.Joy1Caps.wYMax -
      DGCJoystick1.Joy1Caps.wYMin) div 77;
  Panel5.Color := clBlue;
end;
```

En el siguiente procedimiento ajustamos el valor de Threshold. Para ello, utilizaremos el método JoyThresh:

```
procedure TForm1.TrackBar1Change(Sender:
      TObject);
begin
  Label4.Caption := IntToStr(Trackbar1.position);
  DGCJoystick1.Joy1Thresh := Trackbar1.position;
end;
```

Si el usuario ajusta la frecuencia de polling, se llamará automáticamente a este procedimiento y se utilizará el método Joy1Polling:

```
procedure TForm1.TrackBar2Change(Sender:
      TObject);
begin
  Label3.Caption := IntToStr(Trackbar2.position);
  DGCJoystick1.Joy1Polling := Trackbar2.position;
end;
```

Para que el signo + se mueva por su panel, necesitamos encontrar la distancia a la que el joystick debe moverse por cada píxel

Para la gestión de datos enviados por joysticks analógicos, se utilizará este procedimiento. Se utilizarán tres ejes: X, Y, Z (aunque sólo necesitemos los dos primeros). Para colocar el signo + en el panel, tomaremos la posición X del joystick y la dividiremos por el rango calculado anteriormente en la variable XRange. Por ejemplo, si el joystick se encuentra centrado, deberíamos obtener el valor 32768 para nuestra área de representación, ya que suponemos que el valor máximo es 65535. Si 32768 se divide entre nuestro rango (820), obtendremos un valor entre 0 y 80, justo el tamaño de nuestro panel.

```
procedure
      TForm1.DGCJoystick1Joy1AnMove(XPos,
      YPos, ZPos: Integer);
begin
 Label5.Left := XPos div XRange;
 Label5.Top  := YPos div YRange;
end;

end.
```

AHORA EMPIEZA EL TRABAJO...

Y nunca mejor dicho. Durante los 8 artículos que ha durado este curso, hemos visto las bases de los conceptos más fundamentales en la programación de DirectX mediante el potente compilador Delphi. Hemos visto cómo programar sprites, scrolls, joysticks, juegos en red, reproducción de archivos de sonido, etc. Lo único que no se ha tocado de DirectX ha sido la programación 3D, debido a que las librerías del Delphi Games Creator aún no soportan esta materia (esperaremos con impaciencia).

Una vez que se tienen las bases fundamentales para la programación, empieza la tarea más dura: el diseño del juego, gráficos, sonidos, etc. Es en estos aspectos donde más se tarda en la creación de un buen proyecto, sobre todo si se tienen objetivos profesionales.

No desesperéis y darle tiempo al tiempo.

A continuación detallo la dirección de Internet donde se pueden obtener nuevas versiones de las librerías DGC, así como ejemplos:

http://www.users.dircon.co.uk/~zeus/

La página principal de Microsoft sobre DirectX es la siguiente (los ejemplos vienen para C++):

http://www.microsoft.com/directx

PROGRAMACIÓN DE SOCKETS

Programación de WinCGI (IV)

Enrique de la Lastra (elastra@redestb.es)

La interfaz CGI fue la primera en especificar un método de diálogo desde el sentido servidor web hacia el navegador. WinCGI, a su vez, facilita la programación CGI en entornos Wintel. Aquí aprenderemos a crear aplicaciones de servidor que se adapten a dicha interfaz, mediante un programa de ejemplo que demuestre nuestra teoría.

INTRODUCCIÓN

El mes pasado comenzamos a estudiar las posibilidades de la interfaz *WinCGI* que, como vimos, adapta la interfaz *CGI* a los entornos Windows. Fue creada por la empresa de Software *O'Reilly & Associates, Inc* (autores además del servidor web *WebSite)* define el intercambio de datos entre el servidor web y la aplicación *WinCGI*, de forma que este diálogo pueda ser útil entre ambos. En este artículo continuaremos estudiando la interfaz *WinCGI*, aunque esta vez de forma práctica, mediante un programa de prueba que demuestre el funcionamiento del diálogo con un servidor web.

PÁGINA HTML DE LLAMADA AL PROGRAMA WINCGI

Vamos a crear un programa *WinCGI* que acceda a un artículo concreto de los tres escritos hasta ahora en la serie de artículos *Creación de un Servidor web*. Para ello, lo primero que tendremos que hacer es crear un formulario HTML (que será el que tenga una referencia al programa *WinCGI*), en el que se presenten los nombres de los artículos. Cada uno de ellos se encontrará asociado a una casilla de selección. Además, el formulario deberá contener y un botón que active el envío de la petición *POST* al programa *WinCGI* (**cgi_01.exe**). Dicho programa, junto con el código necesario para la comprensión del ejemplo, se encuentra en el CD-ROM de la revista. El código completo de esta página HTML se muestra en el Listado 1 y la apariencia en pantalla se observa en la figura 4 (la explicación del código HTML queda fuera del alcance de este artículo).

NUESTRO PRIMER PROGRAMA WINCGI

Para programar la aplicación *WinCGI*, abrimos un proyecto normal en Delphi. Y lo guardamos con el nombre **cgi_01.dpr** y la *unit* como **cgi_unit.pas**. El programa deberá realizar el procesamiento deseado y una vez terminado, finalizar antes de que se cree el formulario. Para ello, deberemos de insertar una línea en la que se informe al programa que pare la ejecución al terminar la creación del formulario; y debemos eliminar otra línea para evitar la creación de la ventana del formulario.

La línea que se debe insertar consiste en una instrucción Halt que deberá ir al final del método de creación del formulario, como se aprecia en el Listado 2. La línea que debemos eliminar será: Application.Run en el código del proyecto. Es decir, el proyecto cgi_01.dpr deberá quedar de la siguiente manera:

```
program Cgi_01;uses  Forms,  cgi_unit in
    'cgi_unit.pas' {FormPruebaCGI01};{$R
    *.RES}begin
    Application.CreateForm(TFormPruebaCGI0
    1, FormPruebaCGI01);end.
```

Estructuraremos el programa en dos procedimientos, los cuáles se ejecutarán una sola vez dentro del manejador del evento FormCreate. El primero de ellos leerá el archivo INI que envía el servidor web como parámetro en la línea de comandos. El segundo escribirá, en el archivo de salida de intercambio, la respuesta de acuerdo a la petición. El primer procedimiento se denomina ObtenerDatosPeticion y el segundo CrearCabeceras. Empezamos (cómo no) por el primero de ellos.

Para acceder al programa WinCGI debemos crear un formulario HTML que envíe los datos al servidor web mediante el comando POST

La información del archivo INI se envía como parámetro en la línea de comandos. Para recoger los parámetros transmitidos de esta manera Delphi incorpora la función ParamStr(), a la que se le debe pasar el número de orden del parámetro introducido en la línea de comandos. En este caso sólo hay un parámetro, por lo que el código para recuperarlo será el siguiente:

```
NombreFicheroIni := ParamStr(1);
```

Listado 1. Código HTML del formulario empleado para enviar datos a la aplicación WinCGI.

```
<HTML>
<HEAD>
  <TITLE>Prueba de CGI. Servidor Web Solop. Enrique de la Lastra</TITLE>
</HEAD>
<BODY>

<H1>Prueba de CGI. Servidor Web Solo Programadores</H1>
<FORM ACTION="/cgi-bin/cgi_01.exe" METHOD="POST">

<P>
<HR WIDTH="100%"></P>
<P>Seleccione el art&iacute;culo que desee obtener:</P>
<OL>
  <LI>Creaci&oacute;n de un Servidor Web 1/3 <INPUT TYPE="radio" NAME="Articulo"
VALUE="ServidorWeb1.htm"></LI>
  <LI>Creaci&oacute;n de un Servidor Web 2/3 <INPUT TYPE="radio" NAME="Articulo"
VALUE="ServidorWeb2.htm"></LI>
  <LI>Creaci&oacute;n de un Servidor Web 3/3 <INPUT TYPE="radio" NAME="Articulo"
VALUE="ServidorWeb3.htm"></LI>
</OL>
<P><HR WIDTH="100%"></P>

<P><INPUT TYPE="SUBMIT" VALUE="Traer el artículo"></FORM></P>

</BODY>
</HTML>
```

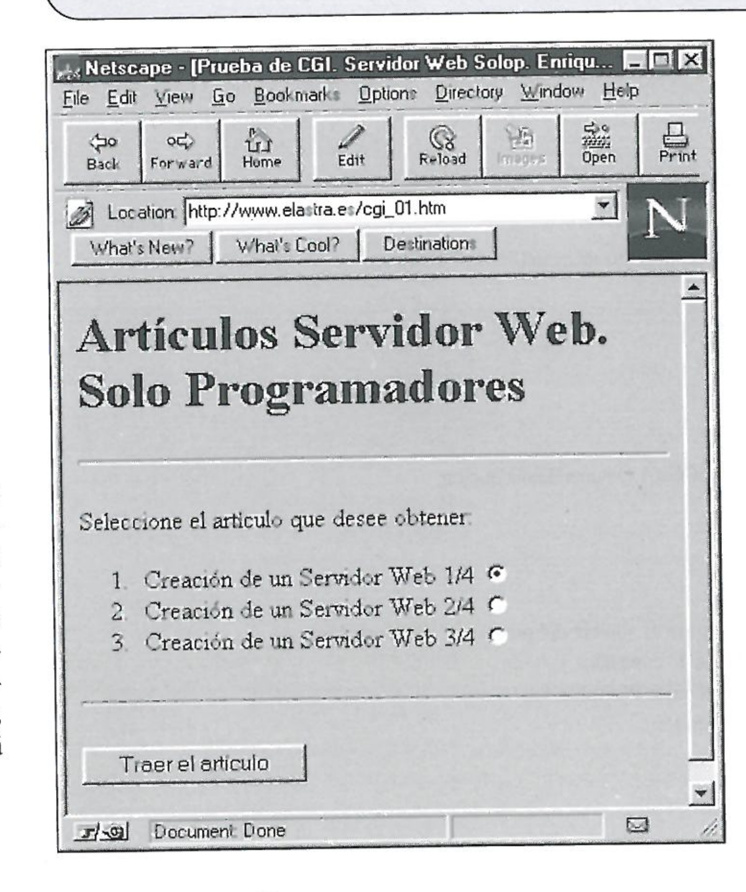


Figura 1. Página HTML desde la que lanzamos la petición al programa WinCGI que hemos creado y que estará residente en el Servidor.

Listado 2. Código completo del programa WinCGI.

```
unit cgi_unit;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, IniFiles;

type
  TFormPruebaCGI01 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    procedure ObtenerDatosPeticion;
    procedure CrearCabeceras (CodigoHTTP: Integer);
  public
    FicheroSalida:      TextFile;
    FicheroPrueba:      TextFile;
    Articulo:    TextFile;
    NombreFicheroSalida: string;

    MetodoPeticion:     string;
    NuestroServidor:    string;
    DireccionCliente:   string;

    ArticuloPedido:     string;
  end;

var
  FormPruebaCGI01: TFormPruebaCGI01;

implementation

{$R *.DFM}

procedure TFormPruebaCGI01.FormCreate(Sender: TObject);
begin
  ObtenerDatosPeticion;
  CrearCabeceras (200);
  Halt;
end;

procedure TFormPruebaCGI01.ObtenerDatosPeticion;
var
  FicheroIni: TIniFile;
  NombreINI: string;
begin
  { El fichero que acompa±a al nombre del programa CGI, serß el primer
    parßmetro en la lØnea de comandos }
  FicheroIni := TIniFile.Create(ParamStr(1));
  NombreINI := ParamStr(1);

  Application.ProcessMessages;
```

Con el nombre del fichero INI en nuestras manos, ya sólo resta abrirlo:

```
FicheroIni := TIniFile.Create(NombreFicheroIni);
```

Y leer desde él las distintas secciones (que nos interesen) de dicho fichero, e irlas almacenando en las variables correspondientes.

La información del archivo ".ini" se envía como parámetro en la línea de comandos

Para leer de un fichero INI disponemos del método ReadString de la clase TIniFile:

Valor_de_la_entrada := FicheroIni.ReadString ('Sección', 'Entrada', 'valor_si_no_encontrado');

Por tanto, para leer, por ejemplo, la entrada Server Software de la sección CGI, escribiremos la línea:

```
NombreServidor := FicheroIni.ReadString ('CGI',
    'Server Software', '');
```

Leemos las secciones y entradas que nos interesen; en nuestro caso (ver el código del procedimiento ObtenerDatosPeticion en el Listado 2), como es un programa de prueba, recogemos casi todas las cabeceras, para después mostrarlas en la página HTML de respuesta que habremos creado.

El procedimiento CrearCabeceras se encargará de establecer las cabeceras de la respuesta HTTP, como si del servidor web se tratara y añadirá el contenido HTML de la respuesta del propio programa WinCGI.

El nombre fichero de salida donde vamos a generar la respuesta está recogido en la variable NombreFicheroSalida que a su vez obtuvimos leyendo la entra-

Si le gustaron otras bases de datos, 4th Dimension v6 *le sorprenderá*

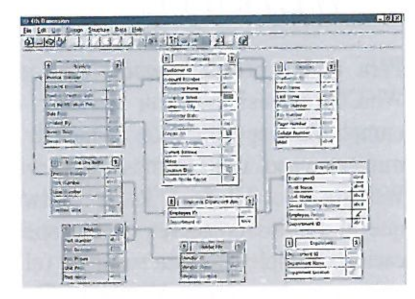


4th Dimension es el entorno de bases de datos integrado más completo del mercado. Años de experiencia en el diseño de bases de datos han permitido satisfacer las siguientes necesidades:

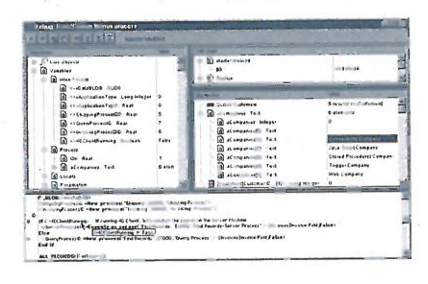
- Automatizar cualquier proceso.
- Diseñar modelos gráficos eficientes y orientados a objeto.
- Suministrar la más amplia gama de herramientas para facilitar el diseño y mantenimiento de bases de datos.
- Permitir una fácil reutilización de código.
- Permitir escalar desarrollos Cliente/Servidor, web y multiplataforma de forma transparente.

Estas son algunas de las **características** de 4th Dimension:

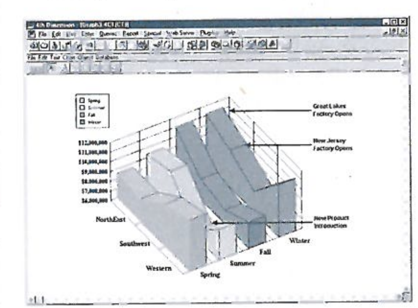
- **Motor de base de datos multiproceso, triggers, procedimientos almacenados**, presentación visual de tablas mediante diagrama de entidad /relación (ERD)



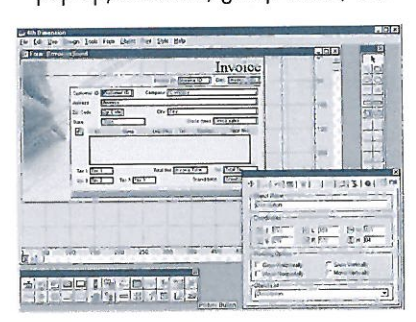
- **Debugger visual** multiproceso de última generación. **Intérprete** incluido para testeo y debugging.



- **Wizards avanzados** para diseño de formularios, queries, informes, importación, exportación, etiquetas, etc. Editor de gráficas programable 2D y 3D, soporta gráficas de barras, tartas, áreas, líneas, etc.



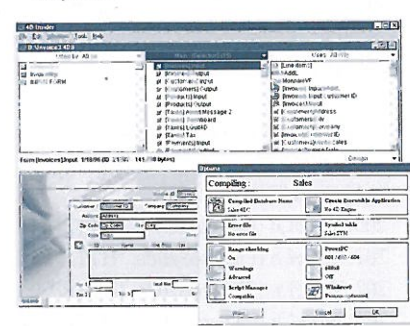
- **Editor de formularios basado en objetos.** Gestión de hojas de estilo. Redimensionamiento automático. **Docenas de objetos de interface** incluyendo check boxes, radio buttons, push buttons, áreas de scroll, picture button, botones invisibles, tab control, listas jerárquicas, termóme-tros, reglas, combo box, listas menú desplegables, menús pop-up jerárquicos, menús picture, radio pictures, listas desplegables, menús pop-up, subforms, group boxes, etc.



- Arquitectura de distribución de aplicaciones integrada y **soporte para proyectos con múltiples desarrolladores** con check-in/check-out automatizado.
- Arquitectura extensible mediante **plug-ins** y DLLs.
- **Un sólo código fuente** desplegable sin modificaciones de monousuario a 4D Server, la versión Cliente/Servidor de 4th Dimension.
- Soporte de **ODBC** (cliente y servidor) y conexiones nativas **SQL** a ORACLE, Sybase y MS SQL Server.
- **Servidor Web integrado** para acceso automático a formularios y gestión Cliente/Servidor mediante navegadores web.



- **Compilador a código máquina real** optimizado para diferentes procesadores. **Gestor gráfico de referencias cruzadas** y librerías de objetos.



VERSIÓN DEMO GRATUITA

4th Dimension es el núcleo de la gama de productos 4D. Visite http://www.ambitconsulting.com o llame al 93-209 41 11 para más información o para recibir una versión demo gratuita. Mencione este anuncio y reciba 4th Dimension v6 por el precio promocional de 34.900 Ptas*. (precio habitual: 79.000 Ptas.).

(* Actualización competitiva. Oferta válida hasta el final de existencias)

Balmes, 297, 4º, 2ªb. 08006 Barcelona
t. | (34) 93 209 41 11 f. | (34) 93 240 18 88
e-mail: ambit@ambitconsulting.com
http://www.ambitconsulting.com



Listado 2. (Continuación).

```
    AssignFile (FicheroPrueba, 'c:\home\temp\salida_cgi.out');
    ReWrite (FicheroPrueba);
    WriteLn (FicheroPrueba, 'Ini: ' + NombreINI);
    CloseFile (FicheroPrueba);

    { Recogemos los datos del fichero ".ini" }
    NombreFicheroSalida := FicheroIni.ReadString('System', 'Output File', '');
    MetodoPeticion   := FicheroIni.ReadString ('CGI', 'Request Method', '');
    NuestroServidor  := FicheroIni.ReadString ('CGI', 'Server Name', '');
    DireccionCliente := FicheroIni.ReadString ('CGI', 'Remote Address', '');
    ArticuloPedido   := FicheroIni.ReadString ('Form Literal', 'Articulo', '');

    AssignFile (FicheroPrueba, 'c:\home\temp\salida_cgi.out');
    Append (FicheroPrueba);
    WriteLn (FicheroPrueba, 'ArtØculo: ' + ArticuloPedido);
    WriteLn (FicheroPrueba, 'M_todo:   ' + MetodoPeticion);
    WriteLn (FicheroPrueba, 'Servidor: ' + NuestroServidor);
    WriteLn (FicheroPrueba, 'Dir. IP:  ' + DireccionCliente);
    WriteLn (FicheroPrueba, 'Fich. Out:' + NombreFicheroSalida);
    CloseFile (FicheroPrueba);

    FicheroIni.Free;
end;

procedure TFormPruebaCGI01.CrearCabeceras (CodigoHTTP: Integer);
var
    s: string;
begin
    AssignFile (FicheroSalida, NombreFicheroSalida);
    ReWrite (FicheroSalida);

    { Creamos el c≤digo de respuesta, tal y como viene definido en
     la especificaci≤n HTTP/1.0 - rfc1945 }
    case CodigoHTTP of
      200: WriteLn (FicheroSalida, 'HTTP/1.0 200 OK');
      201: WriteLn (FicheroSalida, 'HTTP/1.0 201 Created');
      202: WriteLn (FicheroSalida, 'HTTP/1.0 202 Accepted');
      204: WriteLn (FicheroSalida, 'HTTP/1.0 204 No Content');
      301: WriteLn (FicheroSalida, 'HTTP/1.0 301 Moved Permanently');
      302: WriteLn (FicheroSalida, 'HTTP/1.0 302 Moved Temporarily');
      304: WriteLn (FicheroSalida, 'HTTP/1.0 304 Not Modified');
      400: WriteLn (FicheroSalida, 'HTTP/1.0 400 Bad Request');
      401: WriteLn (FicheroSalida, 'HTTP/1.0 401 Unauthorized');
      403: WriteLn (FicheroSalida, 'HTTP/1.0 403 Forbidden');
      404: WriteLn (FicheroSalida, 'HTTP/1.0 404 Not Found');
      500: WriteLn (FicheroSalida, 'HTTP/1.0 500 Internal Server Error');
      501: WriteLn (FicheroSalida, 'HTTP/1.0 501 Not Implemented');
      502: WriteLn (FicheroSalida, 'HTTP/1.0 502 Bad Gateway');
      503: WriteLn (FicheroSalida, 'HTTP/1.0 503 Service Unavailable');
    end;
    { Siguiente lØnea: Fecha y hora }
```

da Output File de la sección System (ver ObtenerDatosPeticion en el Listado 2).

Para crear las cabeceras, reutilizamos el código que empleamos en el servidor web para la misma tarea (ver el segundo artículo sobre esta serie), efectuando una pequeña modificación: sustituir la variable en la que almacenábamos las cabeceras, por una escritura directa de estas cabeceras en el fichero de salida. Generamos el código HTML de la respuesta, utilizando la información de las variables obtenidas en el procedimiento ObtenerDatosPeticion.

Como nota destacada sobre de la información generada en la respuesta, se crean dos enlaces en esta página "dinámica", uno hacia el artículo solicitado y otro hacia la página que llamó al programa WinCGI.

SERVIDOR WEBSITE

Para poder probar nuestra aplicación WinCGI necesitamos un servidor que cumpla la especificación WinCGI. En nuestro caso utilizaremos el servidor WebSite Pro 2.0 de O'Reilly Software, que al ser los creadores de la especificación - cumple con este requisito (el popular servidor web de Microsoft "Personal web Server" no funciona con programas WinCGI).

El servidor WebSite se puede encontrar en el CD-ROM que se regaló con el Especial Monográfico Nº 2 de la revista que se publicó. También se puede encontrar en la dirección web http://-website.ora.com, donde, además del software existe gran cantidad de documentación sobre el servidor.

El ejecutable de instalación del servidor es el archivo wsp2demo.exe. Al ejecutarlo aparecerá una ventana pidiendo el nombre de un directorio temporal donde descomprimir los archivos (por

defecto: c:\cdinter\website). Una vez terminado el proceso de descompresión, y suponiendo que se eligen los directorios por defecto, tendremos un archivo denominado WS2Peval.exe en el directorio: c:\cdinter\website\. Éste es el verdadero programa de instalación del servidor WebSite.

Para probar una aplicación WinCGI, utilizamos WebSite (PWS no cumple la especificación)

Para comenzar la ejecución del programa utilizaremos el archivo WS2Peval.exe, de tal forma que irán apareciendo una serie de ventanas donde se configuran los directorios de instalación y otras opciones. Una de las ventanas que aparecerá, pide que introduzcamos el nombre de dominio que queremos asignar al servidor web y la dirección e-mail de contacto. Ambos, los podemos configurar con los valores que nos resulte conveniente, según nuestras necesidades, teniendo en cuenta que todas las peticiones dirigidas a las direcciones que ahí escribamos, irán encaminadas a nuestro servidor (siempre y cuando esté registrado el nombre en el InterNIC). A partir de este momento, continuará la instalación, tal y como se muestra en la figura 2.

En un principio las peticiones dirigidas al nombre de dominio que hayamos escrito en la configuración, "saldrán" fuera de nuestro PC hacia Internet (lo cual no nos interesa, salvo en el caso de que tengamos un PC conectado directamente a Internet y hayamos adquirido nuestro propio dominio). Para que estas peticiones se dirijan al servidor web que tenemos en local debemos añadir una línea al fichero "Hosts" que se encuentra en el directorio donde se tenga instalado Windows 95. En esta línea, debemos indicarle al PC que las peticiones dirigidas hacia el servidor http://www.elastra.es/ (en este ejemplo) vayan en realidad dirigidas a la dirección IP 127.0.0.1 (es decir, al host local), tal y como se muestra en la figura 3.

Listado 2. (Continuación).

```
WriteLn (FicheroSalida, 'Date: '+ FormatDateTime('ddd, d mmm yyyy hh:mm:ss
"GMT"',Now()));
  { Siguiente l∅nea: Versi≤n MIME }
  WriteLn (FicheroSalida, 'MIME-Version: 1.0');
  { Siguiente l∅nea: Nombre de nuestro Servidor }
  WriteLn (FicheroSalida, 'Server: Servidor Web Solo Programadores v0.1b');

  { Enviamos las cabeceras de contenido, salvo que no exista el recurso, en
    cuyo caso enviamos el nombre del fichero }
  if CodigoHTTP = 200 then begin
    { Comandos de petici≤n permitidos }
    WriteLn (FicheroSalida, 'Allow: GET, HEAD, POST');

    { Content-Type }
    WriteLn (FicheroSalida, 'Content-Type: text/html');
  end;
  WriteLn (FicheroSalida, '');

  WriteLn (FicheroSalida, '<HTML>');
  WriteLn (FicheroSalida, '<HEAD><TITLE>Solo Programadores. Ejemplo
CGI</TITLE></HEAD>');
{ WriteLn (FicheroSalida, '<BODY>Art∅culo: ' + ArticuloPedido);
  WriteLn (FicheroSalida, '<P>M_todo:  ' + MetodoPeticion + '</P>');
  WriteLn (FicheroSalida, '<P>Servidor: ' + NuestroServidor + '</P>');
  WriteLn (FicheroSalida, '<P>Dir. IP:  ' + DireccionCliente + '</P>');
  WriteLn (FicheroSalida, '<P>Fich. Out:' + NombreFicheroSalida + '</P>');
  WriteLn (FicheroSalida, '<P>Fich. Ini:' + NombreIni  + '</P></BODY>');}
  WriteLn (FicheroSalida, '<BODY>Hola</BODY>');
  WriteLn (FicheroSalida, '</HTML>');
  WriteLn (FicheroSalida, '');

{ AssignFile (Articulo, 'c:\home\articulo\ServidorWeb\' + ArticuloPedido );
  Reset (Articulo);
  while not Eof(Articulo) do begin
    ReadLn (Articulo, s);
    WriteLn (FicheroSalida, s);
  end;
  CloseFile (Articulo);}
  CloseFile (FicheroSalida);
end;


end.
```

Una vez hecho esto, ya tenemos el Servidor WebSite listo para funcionar. Para probar la aplicación que hemos desarrollado en el artículo, hay que copiar el programa cgi_01.exe al directorio c:\web-

site\cgi-win\. Después copiar el archivo cgi_01.htm y el directorio \articulo\ServidorWeb\ al directorio c:\website\htdocs\.

Y por último lo que nos falta es lanzar la siguiente petición (es decir, escribimos el siguiente URL) desde cualquier navegador (Netscape Navigator o Internet Explorer):

http://www.elastra.es/cgi_01.htm

En el navegador deberá hacer su aparición en la pantalla tal y como lo vemos en la figura 1.

Si nos proponemos seleccionar, por ejemplo, el artículo número 2, se arrancará en el servidor el programa cgi_01.exe y una vez terminado, el navegador mostrará la pantalla de la dor web se dirijan al PC local.

En esta pantalla, y gracias a la información enviada en la propia petición, tenemos dos enlaces disponibles: uno al artículo solicitado y otro a la página desde la que accedimos a la actual.

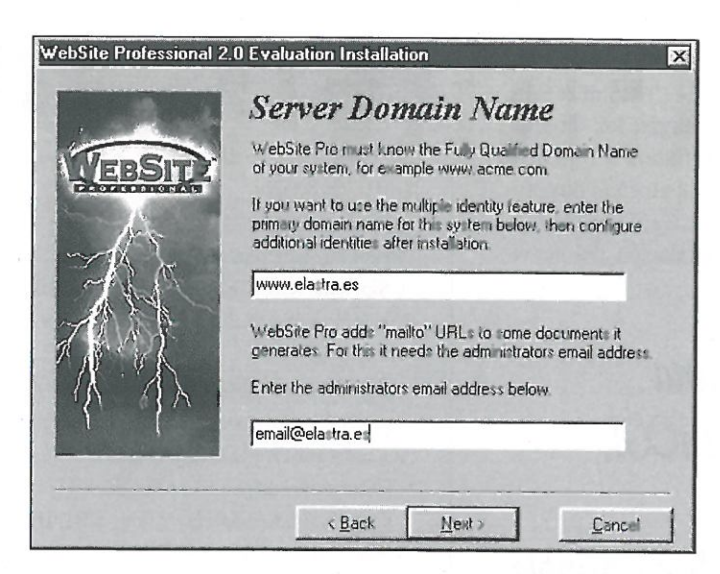


Figura 4. Configuración del Nombre de Dominio y de la dirección e-mail de contacto del servidor web WebSite Pro.

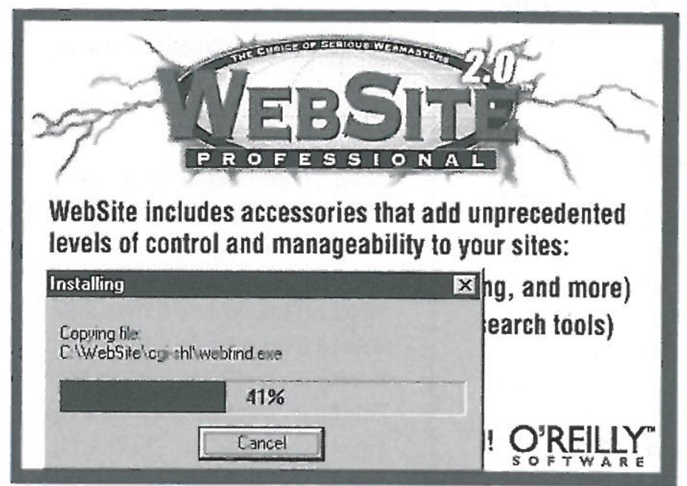


Figura2. Instalación de WebSite Pro 2.0.

UTILIZAR WINCGI CON JAVA WEB SERVER

Para quien esté interesado en el tema y utilice como Servidor web, el "Java Web Server", en la dirección web:

http://www.lib.ksu.edu/tim/servlets/WinCGI.sht

Todos los lectores que acudan a la dirección remitida podrán encontrar el WinCGI Servlet que actúa como si se tratase de una pasarela que permite la posibilidad de ejecutar cualquier programa WinCGI con el servidor web Java Web Server.

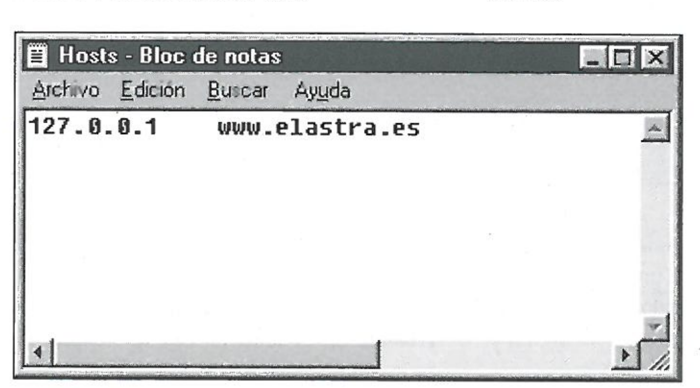


Figura 3. Línea que debemos introducir en el fichero "hosts" de Windows, para que las peticiones HTTP a nuestro Servidor web se dirijan al PC local.

CONCLUSIÓN

Tras seguir las indicaciones hechas a lo largo del artículo, hemos logrado crear un programa WinCGI que será capaz de funcionar con cualquier servidor que sea compatible con la especificación Windows CGI. Además, hemos podido comprobar su correcto funcionamiento en el servidor WebSite.

Durante el próximo artículo, con el que daremos por finalizada la serie dedicada a la creación de un servidor web con Delphi, añadiremos todos los mecanismos necesarios que aun nos faltan para que el servidor pueda dialogar con programas CGI a través de la especificación WinCGI. Para ello daremos soporte al comando POST de HTTP, que permite al cliente enviar los datos de vuelta hacia el servidor web.

Reconocimiento de voz (I)

Constantino Sánchez Ballesteros (constantino@redestb.es)

En este artículo, y los sucesivos, vamos a adentrarnos en la programación de una de las tecnologías más impactantes que puede implementarse a un PC: el reconocimiento de voz. Cualquier desarrollador habrá soñado alguna vez con la idea de que su programa se comunique directamente con el usuario por medio de la voz.

INTRODUCCIÓN

Con la introducción en el mercado de los potentes microprocesadores con tecnología MMX, el hecho de comunicarnos con un procesador ya no es una utopía, y mediante una serie de herramientas, podremos crear programas en los que la interactuación del usuario y la máquina sea lo más cercana posible a la relación común que tenemos con las demás personas.

Aprenderemos a programar la API Speech de Microsoft, encargada de dar soporte de voz a nuestros programas mediante fáciles implementaciones. Los listados de ejemplos que vayamos viendo a lo largo de los artículos estarán escritos en C++ y Visual Basic para tener un mayor rango de posibles programadores interesados en esta tecnología. La programación en Visual Basic para la API Speech es algo menos potente que la de C++, pero no por ello menos flexible.

Por supuesto que todavía falta algún tiempo para que la persona pueda comunicarse de forma fluida con el ordenador al igual que la relación que se mantien entre individuos, pero poco a poco se va logrando esta meta. ¿Cuántas personas creyeron en su momento, que el hombre podría pisar la luna?.

En teoría, el reconocimiento de voz nunca escucha conversaciones entre personas, y cuando se da un comando por parte del usuario, el ordenador nunca comete un error. Si los ordenadores tuvieran un reconocimiento de voz perfecto, todos estaríamos hablando con ellos de forma natural. Desgraciadamente, pasarán décadas antes de que esta tecnología sea perfecta. A pesar de este detalle, el reconocimiento de voz es muy útil en muchas aplicaciones.

Aunque algunas personas esperen algo más de esta tecnología que lo que puede dar de sí, desgraciadamente, la siguen despreciando por completo. En las siguientes líneas se expondrán razones del porqué utilizar el reconocimiento de voz aunque no sea perfecto al cien por cien en la actualidad.

¿POR QUÉ UTILIZAR ESTA TECNOLOGÍA SI NO ES PERFECTA?

Los vídeos basados en ordenadores se introdujeron en 1991. Era una idea realmente innovadora, pero cuando aparecieron los primeros vídeos, se apreció que tenían un tamaño muy limitado. Un ordenador 386/33 Megahercios podía visualizar un vídeo de aproximadamente 80x50

pixels a 15 frames por segundo (fps) y sonido de baja calidad. Desde entonces, la reproducción de vídeos por ordenador ha mejorado significativamente. Actualmente, se visualizan a 320x200 pixels con 24 fps; una cantidad muy superior a la ofrecida en los ordenadores 386 comentados anteriormente. Estas mejoras en resolución y fps todavía son pequeñas en comparación con lo último en tecnología visual: el sonido Dolby Surround digital. Los cines tienen aproximadamente 4000x2000 pixel de resolución, 24-bit color, 4 canales de 44 kHz, y 16 bit de audio. Esta cantidad de datos es varios cientos de veces mayor que la cantidad que somos capaces de visualizar actualmente en el PC de hoy.

El reconocimiento está reemplazando al teclado

Aunque los vídeos por ordenador son significativamente inferiores a las películas, muchas aplicaciones se benefician de la tecnología de vídeo para aplicarla a vídeo-juegos y títulos multimedia. Los vídeos son tan necesarios en algunas aplicaciones que no podrían funcionar sin ellos; aunque estos vídeos estén a años luz de una película real visualizada en el cine.

El reconocimiento de voz y el texto hablado son nuevas tecnologías, tal y como lo son los vídeos en el PC. Aunque las tecnologías del reconocimiento no son todo lo buenas que cabría esperar, muchas aplicaciones pueden obtener rendimientos excelentes utilizándolas.

¿ADIÓS AL TECLADO?

La pregunta es: ¿si el reconocimiento de voz no es perfecto, por qué debería utilizarlo en las aplicaciones?. Después de todo, el reconocimiento está reemplazando al teclado y el ratón, y el teclado nunca comete un error.

No hay que preocuparse. El reconocimiento de voz no reemplazará al teclado y el ratón, ni el texto hablado eliminará el texto escrito. Estas tecnologías únicamente permiten introducir nuevas interfaces de usuario para un mejor manejo del entorno informático. El reconocimiento de voz puede agregarse a una larga lista de dispositivos de interfaces de usuario como el teclado, ratón, joystick o lápiz óptico. Igualmente, el texto hablado puede agregarse al texto, gráficos, vídeos animados, y sonido. Muy pocas aplicaciones utilizarán la voz como único medio de comunicación con los usuarios. La mayoría mezclará y emparejará los dispositivos según sus funcionalidades.

Mediante la API 'Speech' de Microsoft, podremos crear programas que entiendan al usuario con un alto porcentaje de aciertos

Si echamos un vistazo al mercado, muchas aplicaciones utilizan dispositivos de interfaces de usuario para comunicarse con la persona; un juego hace uso del joystick y teclado (a veces, también el ratón). Los usuarios manipulan el joystick y le dicen al ordenador en qué dirección se va a mover el personaje del juego. El teclado se usa para teclear órdenes como "Coger objeto". El joystick es un dispositivo de entrada mucho mejor para el movimiento de objetos por la pantalla, mientras que el teclado es mejor para introducir texto. De forma análoga, el reconocimiento de voz es simplemente otro "dispositivo" que puede combinarse con los existentes para ampliar las funcionalidades de una aplicación.

¿QUÉ PUEDE HACER EL RECONOCIMIENTO DE VOZ?

Después de haber expuesto las razones por las que es posible incluir la tecnología del reconocimiento en nuestros programas, pasaremos a detallar las capacidades actuales que tiene el texto hablado y el reconocimiento de voz.

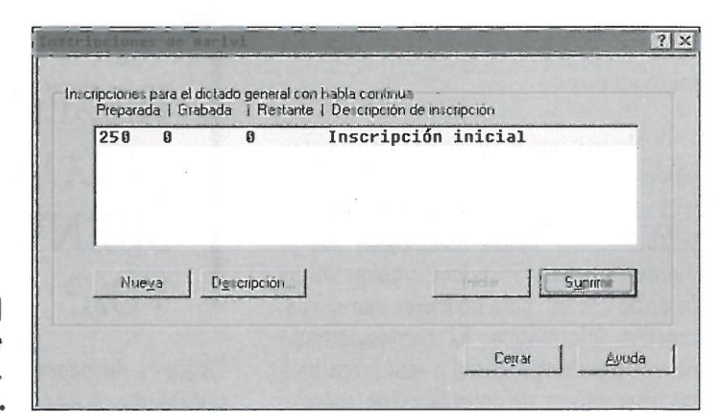
El texto hablado tiene 2 vertientes:

1. Sintetizada
2. Encadenada

La voz sintetizada es lo que la gente suele pensar cuando se comenta el detalle "texto hablado". Este sistema lee el texto analizando las palabras sin tener en cuenta la pronunciación fonética. Los fonemas se pasan a un algoritmo complejo que simula el tracto vocal humano y emite el sonido. Este método permite hablar cualquier palabra, por muy extraña que sea, pero produce una voz que tiene una emoción muy pequeña y no es humana. Podríamos utilizar este método en nuestra aplicación si no necesitamos predecir lo que se tiene que decir. La voz sintetizada normalmente requiere de un 486/33 Megahercios con 1 megabyte para el trabajo de RAM.

La voz encadenada hace algo diferente. Analiza el texto y ejecuta grabaciones, palabras, y frases de una biblioteca pre-grabada. Seguidamente, las grabaciones de audio digital se encadenan. Dado que la voz es una grabación que hemos hecho, este sistema parece eficiente. Desgraciadamente, si el texto incluye una palabra o frase que no grabamos con anterioridad, el engine del texto hablado no podrá decirla.

El método encadenado puede verse como una forma de condensación de audio, ya que las palabras o frases comunes



Entrenamiento del Reconocimiento de Voz en Viavoice.

sólo tienen que grabarse una sola vez. Por ejemplo, muchas aplicaciones telefónicas tendrán una grabación para, "Pulse 1 para reproducir el mensaje nuevo; Pulse 2 para enviar un fax...", y otra para, "Pulse 1 para pasar de mensaje; Pulse 2 para rebobinar". Un texto hablado encadenado sólo tendrá una grabación de "Pulse" en lugar de cuatro. Mediante este sistema, nos ahorramos espacio de almacenamiento y se obtiene un mejor rendimiento del sistema. No será necesario un ordenador muy potente puesto que la mayoría de los datos de audio grabados permanecerán en el disco duro para cuando se quieran reproducir.

Todavía no se ha creado el engine de reconocimiento perfecto, y aún pasará bastante tiempo

De este momento en adelante, tomaremos la palabra SPEAKER para hacer referencia a la "persona que habla al ordenador".

El reconocimiento de voz se complica un poco más a la hora de clasificarlo en relación con el texto hablado. Cada engine de reconocimiento de voz tiene tres características:

1. *Continuo y Discreto*: Si el reconocimiento de voz es continuo, los usuarios pueden hablar de forma natural al ordenador. Si es discreto, los usuarios necesitan hacer una pausa entre palabras. Obviamente, se prefiere el reconocimiento continuo en lugar del discreto, con la salvedad de que el continuo requiere más tiempo de proceso para el PC.

2. *Tamaño del vocabulario*: El reconocimiento de voz puede incluir un vocabulario pequeño o grande. Un vocabulario pequeño permite a los usuarios dar órdenes simples a sus ordenadores. Para poder dictar un documento, el sistema debe tener un vocabulario grande para el reconocimiento, aumentando los requisitos de memoria y procesador.

3. *Dependencia del speaker*: El reconocimiento de voz con independencia del speaker, trabaja directamente sin necesidad de entrenamiento alguno; por el contrario, en sistemas con dependencia del speaker se requiere que cada usuario utilice aproximadamente 30 minutos para entrenar al sistema con su propia voz.

Aunque cualquier combinación de las tres características es posible, dos de ellas son las más utilizadas actualmente:

1. "Command and Control" (comando y control): este tipo de reconocimiento de voz es continuo, utiliza un vocabulario pequeño y un speaker independiente. Esto significa que los usuarios pueden utilizar varios centenares de órdenes diferentes o frases. Si un usuario dice una orden que no está en la lista, el sistema del reconocimiento de voz devolverá la frase "Comando no reconocido", o pensará que oyó un comando sonoro similar. Dado que los usuarios de "Command and Control" sólo pueden decir frases específicas, éstas deben ser visibles en pantalla para que el usuario sepa qué decir en cualquier momento.

Command and Control utiliza un speaker independiente

2. "Discrete Dictation" (dictado discreto): utiliza un vocabulario discreto, grande, y dependiente del speaker. Se utiliza para dictar en los procesadores de texto, mandar correo electrónico, o dictar comandos naturales del lenguaje que se utilice. Aunque los usuarios puedan decir lo que deseen, deben dejar pausas entre las palabras y realizar el dictado de forma antinatural, es decir, que lo que se hable no sonará de igual forma que cuando lo hacemos con otras personas, debido a las pausas efectuadas. El dictado discreto requiere un Pentium/60 con 8 megabytes de RAM (y quizá sea poco).

EJEMPLOS DE LA UTILIZACIÓN DEL RECONOCIMIENTO Y EL HABLA

Hay que tener en mente que cada aplicación es diferente y efectúa tareas de diversa índole. Como regla general, se debe uti-

lizar el reconocimiento de voz cuando proporcione ventajas significantes a los usuarios, no por el hecho de introducir esta tecnología sin ningún objetivo claro. Hay que encontrar usos que hagan a las aplicaciones más fáciles de utilizar, o permitan a los usuarios realizar tareas de forma más rápida de lo que lo harían con el ratón o el teclado. En el caso de los vídeojuegos, se puede utilizar la voz para agregar realismo en la interacción con los objetos del juego y permitir más diversión.

Las aplicaciones de la telefonía se benefician en gran medida del reconocimiento y el habla porque el único medio de comunicarse con el usuario es a través del teléfono. El texto hablado no sólo es más fácil de utilizar que la grabación manual, sino que además permite nuevas características como lectura de texto arbitrario (nombres, direcciones, números, etc.). El reconocimiento de voz es un gran sustituto de los menús realizados por los tonos de los botones del teléfono, no sólo por su naturalidad y flexibilidad, sino porque muchos usuarios no tienen teléfonos que permitan la funcionalidad anteriormente comentada.

Una aplicación que permita reconocimiento de voz, debe informar al usuario de las posibilidades de uso

Los juegos, sobre todo las aventuras gráficas, son los próximos beneficiarios de la voz. Cualquier juego en el que el usuario "hable" con los personajes ganará muchísimos enteros. El reconocimiento de voz permite a los usuarios hablar realmente con los personajes de una forma más divertida que si tuviéramos que teclear cada contestación. En lugar de visualizar (de forma escrita) los mensajes que dicen los personajes, utilizando el texto hablado haríamos que hablaran con nosotros como si fueran personas. Todos estos detalles aumentan el realismo del juego, y además permite a los usuarios quitar las manos del teclado y disfrutar del juego sin centrarse en otras cosas secundarias.

Los títulos multimedia pueden usar texto hablado encadenado para los pasajes de audio. También pueden incorporar reconocimiento de voz para que los usuarios puedan manejar de forma más rápida y eficiente todos los menús de la aplicación sin interferir con el teclado. Por supuesto, hay otras aplicaciones que pueden usar voz; por ejemplo, las que requieran que el usuario tenga las manos libres para efectuar otras tareas, o aquellas funciones que el usuario no pueda ver en pantalla de forma instantánea.

FUNCIONAMIENTO GLOBAL DE LA TECNOLOGÍA DEL RECONOCIMIENTO Y EL HABLA

El reconocimiento de voz es la habilidad de un ordenador para entender la palabra hablada con el propósito de ejecutar una acción concreta con los datos obtenidos. El texto hablado es la habilidad de un ordenador para convertir la información de tipo texto en sonidos que el humano pueda entender, y siempre expresando el texto escrito.

La mayoría de los engines de reconocimiento convierten datos de audio entrantes a fonemas específicos que se traducen en texto para que una aplicación pueda utilizarlo (un fonema es la unidad estructural más pequeña de sonido que puede usarse para distinguir una pronunciación de otro en un idioma hablado). Un engine de texto hablado realiza el mismo proceso pero en orden inverso. Estos son proporcionados por vendedores que se especializan en software de voz.

El engine del reconocimiento de voz transcribe datos de audio recibidos de una fuente específica, como un micrófono o una línea telefónica. El engine de texto hablado convierte texto a datos de audio que se envían a una fuente destino, como un auricular o una línea telefónica. Bajo algunas circunstancias, un engine puede transcribir los datos de audio directamente a un archivo.

El engine mantiene típicamente más de un modo, reconociendo voz o hablando texto. Por ejemplo, un engine de reconocimiento tendrá un modo para cada idioma o dialecto que pueda reconocer. Igualmente, uno de texto hablado tendrá un modo para cada voz que reproduzca un texto hablando en un estilo o personalidad diferente.

El reconocimiento de voz puede ser tan simple como un juego de órdenes de voz predefinidas que una aplicación puede reconocer. Uno más complejo puede involucrar el uso de una gramática que define un juego de palabras y frases que pueden reconocerse. Una gramática puede usar reglas para predecir palabras, probablemente para seguir la palabra hablada, o puede definir un contexto que identifique el asunto de dictado y el estilo esperado de idioma.

Los engines convierten datos de audio entrantes a fonemas específicos que se traducen al texto

Los engines de reconocimiento y habla pueden hacer uso de un léxico de la pronunciación, el cual es un banco de datos de pronunciaciones correctas para las palabras y frases que se han reconocido previamente. La aproximación de

Nombre	Vocabulario	Tipo	Descripción
'	TODAS	macro	Pronunciado apóstrofo
-	TODAS	macro	Pronunciado guión guión-unido
!	TODAS	macro	Pronunciado cerrar-admiración, cerrar-exclam...
"	TODAS	macro	Pronunciado comillas
#	TODAS	macro	Pronunciado signo-número
$	TODAS	macro	Pronunciado signo-dólar
%	TODAS	macro	Pronunciado por-ciento
&	TODAS	macro	Pronunciado ampersand, signo-i
(	TODAS	macro	Pronunciado abrir-paréntesis
)	TODAS	macro	Pronunciado cerrar-paréntesis
*	TODAS	macro	Pronunciado asterisco
,	TODAS	macro	Pronunciado coma
.	TODAS	macro	Pronunciado punto, punto-y-seguido
...	TODAS	macro	Pronunciado puntos-suspensivos
/	TODAS	macro	Pronunciado barra

Parámetros de habla
Nombre del usuario: ugre — Vocabulario: Vocabulario General para ViaVoice

El uso de macros en Viavoice permite mayor riqueza en el dictado.

un engine para reconocer voz o reproducir texto determina la calidad de voz en una aplicación; es decir, la exactitud de reconocimiento o claridad de reproducción y la cantidad de esfuerzo requerido por parte del usuario para conseguir una buena exactitud o claridad. La exactitud que emplee el engine también afecta a la velocidad del procesador y la memoria requerida por una aplicación.

RECONOCIMIENTO DE VOZ

Todo reconocimiento de voz involucra el descubrir y reconocer nuevas palabras. La mayoría de este tipo de engines pueden categorizarse dependiendo de las tareas básicas que realicen en las siguientes categorías:

- La separación: el grado de aislamiento entre palabras requeridas por el engine para reconocer una palabra.
- La dependencia del speaker: el grado al que se restringe el engine dependiendo de un determinado speaker.
- Técnicas de emparejamiento: método que utiliza para emparejar una palabra descubierta con las conocidas del vocabulario.
- Tamaño de vocabulario: número de palabras que el engine investiga para emparejar una palabra.

Los engines de reconocimiento de voz suelen exigir uno de los siguientes tipos de entrada verbal para descubrir las palabras nuevas:

- Voz discreta: cada palabra debe ser aislada por una pausa antes y después de la palabra en sí (normalmente sobre un cuarto de segundo) para que el engine pueda reconocerla. El reconocimiento discreto requiere mucho menos proceso que el continuo, pero es más incómodo para el usuario.

Speech soporta dos módulos diferenciados: reconocimiento de voz y texto hablado

- Palabra determinada: Una serie de palabras puede hablarse en una pronunciación continua, pero el engine sólo reconoce una palabra o frase determinada. Por ejemplo, si el engine usa la palabra "tiempo" y el usuario ha dicho "qué tiempo hace" o "tiempo de revancha", sólo se reconocerá la palabra "tiempo." Este método se utiliza cuando se espera del usuario un número limitado de órdenes o respuestas y la manera en que el usuario habla es imprevisible o insignificante.
- Voz continua: el engine encuentra una pronunciación continua sin necesidad de establecer pausas entre las palabras, y puede reconocer todas las palabras que se hablaron. El reconocimiento de voz continuo es la mejor tecnología desde un punto de vista útil, porque es el estilo hablado más natural para los seres humanos. Sin embargo, el costo de proceso es intensivo porque la identificación del principio y final de todas las palabras es bastante difícil (a mucha gente le gusta leer texto impreso sin espacios o puntuación).

DEPENDENCIA EN EL SPEAKER

Los engines del reconocimiento de voz pueden requerir un entrenamiento previo para reconocer bien el habla de un usuario en particular, o que éstos se adapten en gran medida al engine. En este campo, los engines pueden agruparse en las siguientes categorías:

La dependencia del speaker es el grado al que se restringe el engine

- Dependiente del speaker: el artefacto le exige al usuario que lo entrene para reconocer su voz. Normalmente, el entrenamiento involucra el habla de una serie de frases preseleccionadas. Cada nuevo speaker debe realizar el mismo entrenamiento. Estos engines pueden trabajar sin entrenarlos, pero su exacti-

tud normalmente empieza por debajo del 95 por ciento y no mejora hasta que el usuario completa el entrenamiento. Esta técnica toma la menor cantidad de proceso, pero está limitada para la mayoría de los usuarios porque el entrenamiento es tedioso y suele variar de 15 minutos a varias horas.

El usuario debe informar al engine de algún modo cuando cometa un error para que no aprenda basándose en éstos

- Adaptable al speaker: el engine se entrena para reconocer la voz del usuario mientras éste realiza tareas ordinarias. La exactitud normalmente empieza aproximadamente a un 90 por ciento, pero se eleva a unos niveles más aceptables después de unas horas de uso. Deben tenerse en cuenta dos consideraciones con esta tecnología:

    1. El usuario debe informar al engine de algún modo cuando cometa un error, para que éste no aprenda basándose en esos errores.

    2. Aunque el reconocimiento mejora para el usuario individual, otras personas que intenten usar el sistema conseguirán una tasa de error más alta hasta que hayan uitilizado el sistema durante algún tiempo.

- Independiente del speaker: el engine empieza con una exactitud de un 95 por ciento para la mayoría de los usuarios (aquéllos que hablan sin acentos de provincia). Casi todos estos engines tienen entrenamientos o habilidades adaptables por las que mejoran su exactitud en unos porcentajes más altos, pero no requieren su uso. Los sistemas independientes del speaker necesitan más tiempo de proceso que los dependientes.

TÉCNICAS DE EMPAREJAMIENTO

Los engines de reconocimiento de voz emparejan una palabra descubierta con una palabra conocida mediante el uso de las siguientes técnicas:

- Emparejado de palabra entera: el engine compara el audio digital entrante con una plantilla previamente grabada que contiene esa palabra. Esta técnica toma mucho menos proceso que el emparejado de sub-palabras, pero requiere que el usuario (o alguien determinado) grabe previamente cada palabra que se vaya a reconocer (varias cientas de miles). Las plantillas de palabras enteras también requieren grandes cantidades de almacenamiento (entre 50 y 512 bytes por palabra) y sólo es práctico si el vocabulario del reconocimiento es conocido cuando se desarrolla la aplicación.

Se compara el audio digital entrante con una plantilla previamente grabada

- Emparejado de sub-palabras: el engine busca sub-palabras (normalmente fonemas) y entonces realiza el reconocimiento del modelo más allá de ellos. Esta técnica toma más proceso que el emparejamiento anterior, pero requiere menos almacenamiento (entre 5 y 20 bytes por palabra). Además, la pronunciación de la palabra puede suponerse del texto en español sin exigirle al usuario que la hablara de antemano.

TAMAÑO DE VOCABULARIO

Todos los engines se suelen apoyar en vocabularios de diferentes tamaños. El tamaño del vocabulario no representa el número total de palabras que un engine dado pueda reconocer. En cambio, determina el número de palabras que éste puede reconocer con precisión en un estado dado que es definido por la palabra o palabras que se hablaron a tiempo antes del punto actual. Por ejemplo, si un engine está escuchando "Me Dice el tiempo", "Me Dice el día", y "Qué año es?", y el usuario ya ha dicho "dice," "me," y "el", el estado actual tiene estas dos palabras: "tiempo" y "día."

Los engines se apoyan en tres tamaños de vocabulario:

- Vocabulario pequeño: el engine puede reconocer con precisión aproximadamente 50 palabras diferentes en un estado dado. Aunque muchos engines con vocabulario pequeño pueden exceder este número, un mayor número de palabras reducen la exactitud del reconocimiento significativamente y aumentan la carga del cómputo global por parte del procesador. Estos engines son aceptables para el "Command and Control", pero no para el dictado.

- Vocabulario elemento: el engine puede reconocer con precisión aproximadamente 1000 palabras diferentes en un estado dado. Se utiliza para la entrada de los datos y el "Command and Control".

- Vocabulario grande: puede reconocer varios miles de palabras en un

estado dado; varios vocabularios que existen en el mercado sobrepasan las 200.000 palabras. Los engines basados en estos vocabularios requieren mucho más cómputo que los dos anteriores, pero son necesarios para el dictado.

RECONOCEDORES DE VOZ EN EL MERCADO

Hay tres tipos:

- Command and Control: permiten a los usuarios dar órdenes simples como "Minimiza Ventana" o "Envía correo a Constantino". También permite la entrada de datos limitada (como los números). Estos engines utilizan habla continua, independencia del speaker y vocabulario pequeño.

- Dictado discreto: permiten a los usuarios hablar texto arbitrario con el tamaño que sea a un ratio de 50 palabras por minuto. Los engines de dictado discretos utilizan voz discreta, adaptable al speaker y un vocabulario grande.

La voz de la tecnología actual tiende a sonar menos humana que otra producida por el encadenamiento

- Dictado continuo: permiten a los usuarios hablar texto arbitrario utilizando voz continua y proporciona un ratio de entrada de 120 palabras por minuto. También utilizan un vocabulario grande.

TEXTO HABLADO

Para dar voz a un texto escrito, un engine de texto hablado debe determinar primero los fonemas exigidos al hablar una palabra y entonces traducir esos fonemas en datos de audio digital. La mayoría se pueden categorizar por el método que ellos acostumbran a traducir fonemas en sonido audible. Estos engines traducen los fonemas a sonidos de varias formas:

El procesamiento de voz permite dar órdenes muy simples y la entrada de datos limitada

- Palabras encadenadas: aunque estos sistemas son realmente sintetizadores, difieren en algo internamente. En estos engines, el diseñador de la aplicación mantiene grabaciones de las frases y palabras proporcionadas por el usuario. El engine "pega" las grabaciones unas con otras para hablar una frase. Por ejemplo, si utilizamos correo por voz (tan usual actualmente), el engine reproducirá algo como esto: " [Usted tiene] [tres] [mensajes nuevos]". El engine tiene grabaciones para "Usted tiene", todos los números, y "mensajes nuevos".

- Síntesis: un engine que utiliza síntesis aplica varios filtros para simular la longitud de la garganta, cavidad de la boca, forma del labio, y posición de la lengua. La voz producida por la tecnología de la síntesis actual tiende a sonar menos humana que una voz producida por el encadenamiento, pero es posible obtener calidades de voz diferentes con sólo cambiar unos pocos parámetros.

- Encadenamiento: unen segmentos de audio para producir un sonido continuo. Cada Segmento consiste de dos fonemas, uno que lleva el sonido y otro que lo termina.

LA API SPEECH (DETALLES TÉCNICOS)

Utiliza objetos COM bajo Win32 (Windows 95/98 y Windows NT). Los objetos pueden ser accedidos a través de C/C++ o Visual Basic OLE Automation. La arquitectura de la API se divide en dos niveles:

1. Alto nivel: diseñado para permitir facilidad de uso y velocidad en la implementación de código.

2. Bajo nivel: que permite a las aplicaciones poseer un control total de la tecnología.

En los siguientes apartados veremos algunos ejemplos de cómo utilizar Speech en C++ utilizando el alto nivel en los comandos de voz y texto hablado.

COMANDOS DE VOZ

La interface de comandos de voz (alto nivel) se utiliza para el reconocimiento de voz. Con esta interface, un usuario envía al PC comandos simples como "Abrir fichero", o puede contestar a preguntas simples de "Si o No". "Command and Control" no permite el dictado; los desarrolladores que deseen implementar dictado en sus programas deberán utilizar la API de bajo nivel.

El diseño de comandos de voz se parece a los menús de Windows, permitiendo un menú de comandos que el usuario puede decir. Básicamente, cuando se uti-

lizan los comandos de voz, una aplicación diseña un menú de voz que se corresponde con la ventana (formulario) de la aplicación. Muchos programas tendrán un menú de voz para el formulario principal y otro para cuadros de diálogo.

La instancia de la clase CMyIVCmdNotifySink recibe las notificaciones

Cada menú de voz contiene una lista de comandos que el usuario puede decir. Cuando éste dice alguno de ellos, la aplicación es notificada con el comando que se haya dicho. "Abrir fichero" y "Enviar mail a <nombre>" son comandos de voz típicos. Cada comando tiene información adicional al comando hablado, tal como una identificación (ID) o una descripción.

EJEMPLO

Como introducción en la programación, vamos a ver un listado de cómo agregar nuevos comandos de voz. El listado está en C++ y es muy simple:

```
CoCreateInstance (CLSID_VCmd, NULL,
      CLSCTX_LOCAL_SERVER,
      IID_IVoiceCmd, &pIVoiceCommand);
CoCreateInstance() //instancia una copia local del
      objeto (Comandos de voz).
pIVCmdNotifySink = new CMyIVCmdNotifySink;
```

Lo primero que se ha hecho es crear una instancia de la clase CMyIVCmdNotifySink que recibe notificaciones cuando el usuario dice un comando. La funcionalidad de la clase es específica a la aplicación:

```
pIVoiceCommand->Register ("",
      pIVCmdNotifySink, IID_IVCmdNotifySink,
      0, NULL);
```

Cuando se llama al método Register, se informa al objeto de comandos de voz en la cola de notificaciones y se especifica de dónde viene el audio.

```
sData.pData = un puntero a una lista de estructu-
      ras VCMDCOMMAND;
sData.dwSize = tamaño del puntero en memoria
      para sData.pData;
```

Seguidamente, la aplicación debe incluir una lista de estructuras VCMDCOMMAND. Cada estructura contiene un comando (por ejemplo, "Abrir fichero"), una descripción ("Abre un fichero almacenado en disco"), una ID del comando para su identificación, y algún otro parámetro menos relevante. Los comandos son específicos para esa aplicación:

```
strcpy (VCmdName.szApplication, "Nombre de la
      aplicación");
strcpy (VCmdName.szState, "Principal");
pIVoiceCommand->MenuCreate (&VCmdName,
      NULL, VCMDMC_CREATE_TEMP,
      &pIVCmdMenu);
pIVCmdMenu->Add (number of commands, sData,
      NULL);
```

El código anterior crea una instancia de un objeto Menú de voz utilizando los comandos especificados en sData. El menú generado en este ejemplo es temporal, aunque pueden guardarse en el sistema de forma similar al registro, para que la aplicación no tenga que reconstruir las estructuras VCMDCOMMAND cada vez que se ejecute el programa.

```
pIVCmdMenu->Activate (hWnd, 0);
```

Cuando se llama al método Activate, se informa al engine de reconocimiento de voz que comience a "escuchar" los comandos especificados siempre que la ventana, reflejada en el parámetro hWnd, esté activa. Una vez que la aplicación ha llamado al método Activate, el reconocimiento de voz empezará a escuchar. Cuando el usuario diga uno de los comandos del menú, la notificación será activada con la instrucción pIVCmdNotifySink->CommandRecognize (..., dwID, ...). dwID contendrá el ID del comando hablado, al igual que un menú típico de Windows. Existen otras notificaciones como el volumen detectado en la voz, pero la mayoría de las aplicaciones se preocupan del comando hablado (dWID).

Cuando una aplicación no requiere más reconocimientos de voz para el menú, se efectuará la siguiente llamada al código para la desactivación del engine:

```
pIVCmdMenu->Deactivate ();
```

Finalmente, cuando la aplicación acaba, se vacía el objeto creado:

```
pIVCmdMenu->Release();
pIVoiceCommand->Release();
```

La espera de la notificación pIVCmdNotifySink es vaciada directamente por el engine.

OBJETOS

En el reconocimiento de voz están involucrados 4 objetos:

- Comandos de voz: este es el punto central de control del módulo comandos de voz. La mayoría de las aplicaciones sólo lo utilizarán para crear objetos de menú de voz o enumeradores (comentados más abajo).

Microsoft Speech soporta dos API's diferenciadas: una de alto nivel y otra de bajo nivel

- Bandeja de notificaciones: es creada por la aplicación. El objeto Comandos de voz llama a los métodos de este objeto cuando se dice algún comando o se actualiza el nivel de volumen de la fuente de audio.

- Menú de voz: provee la mayoría de los controles del reconocimiento de voz y trabaja de forma similar a un

menú de Windows accedido con HMENU. Este objeto permite que los comandos sean agregados o borrados del menú y se active o desactive su reconocimiento.

El texto de voz es una interface de alto nivel para el texto hablado

- Enumerador del menú de voz: el enumerador se utiliza para obtener la lista de menús de voz que están almacenados en la base de datos interna y determina cuál de ellos está activo.

TEXTO DE VOZ

El texto de voz es una interface de alto nivel para el texto hablado. Como se comentó anteriormente, se encarga de hablar cualquier texto escrito. A continuación veremos un pequeño ejemplo de la inicialización que requiere este objeto y su facilidad de manejo:

```
CoCreateInstance (CLSID_VTxt, NULL,
        CLSCTX_LOCAL_SERVER,
        IID_IVoiceText, &pIVoiceText);
```

Cuando se llama a la función CoCreateInstance se crea una instancia del objeto texto de voz:

```
pIVoiceText->Register ("", "Mi Programa", NULL,
        IID_IVTxtNotifySink, NULL, NULL);
```

Todas las aplicaciones deben llamar al método Register para que el objeto sepa el nombre de la aplicación y qué dispositivo de audio utilizará la API para la reproducción del sonido, puesto que algunas aplicaciones lo hacen utilizando el teléfono. Además, una aplicación puede proveer una bandeja de notificaciones para que se alerte cuando comience o pare el habla del texto.

El siguiente paso será enviar el texto para que sea hablado:

```
pIVoiceText->Speak ("Hola a todos los lectores de
        SoloProgramadores.", 0, NULL);
```

Finalmente, cuando la aplicación ha terminado de usar el texto de voz, se vacía el objeto:

```
pIVoiceText->Release();
```

OBJETOS

La programación relacionada con el texto hablado consta de 2 objetos:

- Texto de voz: este es el objeto que maneja la voz, y el único que la aplicación puede utilizar.

- Bandeja de notificaciones: el código para las notificaciones es opcional y lo implementa la aplicación. El objeto texto de voz llama a los métodos de este objeto cuando el audio reproduce o para la voz hablada.

IMPLEMENTACIONES DE LAS INTERFACES A BAJO NIVEL

Generalmente, mucha gente no estará satisfecha con las interfaces de alto nivel porque no tienen demasiada flexibilidad y control sobre el engine. Por consiguiente, la API también posee una interface de bajo nivel que requiere un poco más de trabajo a la hora de programar, pero que permite mayor control sobre el reconocimiento de voz y los procesos del texto hablado. Los módulos de alto nivel (comandos de voz y texto hablado) utilizan esas mismas interfaces de bajo nivel para el reconocimiento.

Reconocimiento a bajo nivel

Cuando una aplicación utiliza las interfaces de bajo nivel, accede directamente al engine. Esto permite a la aplicación mucho más control pero también incrementa la dificultad de programación. Dado que esta API es más compleja, en esta serie de artículos no entraremos en detalles sobre cómo utilizar el engine, pero sí se dará una idea general de su funcionamiento.

La API posee un interface que requiere un poco más de trabajo a la hora de programar

La API de bajo nivel consta de muchos mas objetos que la de alto nivel. A continuación se detalla cómo funciona esta API:

1. La aplicación determina de dónde viene el audio del reconocimiento de voz y crea un objeto de fuente de audio a través del cual el engine adquiere los datos. Microsoft ha implementado un objeto de fuente de audio que toma los sonidos del dispositivo multimedia wave-in, pero la aplicación puede utilizar fuentes de audio personalizadas como adquirir audio de un fichero WAV o un dispositivo hardware especializado.

2. La aplicación, a través del enumerador, localiza un engine de reconocimiento que se pueda utilizar. Seguidamente, crea una instancia del objeto y lo pasa al objeto de fuente de audio.

3. El objeto engine tiene una caja de diálogo con el objeto de fuente de audio. Su misión es encontrar formatos de datos comunes para los datos de audio digital; como la modulación por pulsos (PCM). Una vez establecido un formato, el engine crea una

bandeja de notificaciones para la fuente de audio. A partir de ahí, el objeto de fuente de audio envía datos de audio digital al engine a través de la bandeja de notificaciones.

4. La aplicación puede registrar una notificación principal que reciba gramática independiente, y saber si el usuario está hablando o no.

5. Cuando todo está preparado, la aplicación crea uno o más objetos de gramática. Éstos son similares a los objetos menú de voz de los comandos de voz, pero más flexibles en la sintaxis del reconocimiento.

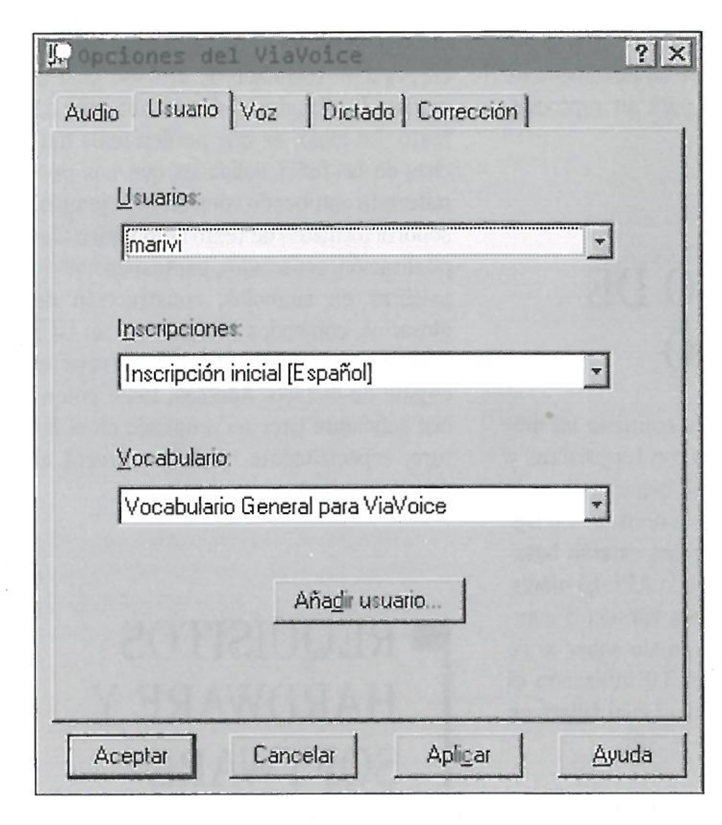
La aplicación pasa uno o más buffers del texto al engine

6. A la hora de encontrar qué palabras ha dicho el usuario, se efectúan los siguientes pasos: la aplicación crea una notificación de gramática por cada objeto de gramática. Cuando el objeto reconoce una palabra o frase, se llama a las funciones de la notificación de gramática. Entonces, la aplicación responde a las notificaciones y toma las acciones que sean necesarias.

7. Típicamente, cuando un objeto de gramática reconoce el habla, envía la notificación a una cadena indicando qué palabra se ha dicho. De todos modos, el engine puede saber más información mediante frases alternativas que se hayan dicho o quién dijo la frase.

Texto hablado a bajo nivel

Cuando una aplicación utiliza esta interface, accede directamente al engine. Al igual que en el reconocimiento de voz, se tiene mucho más control, pero se requiere más programación.



Opciones del ViaVoice para modificar el reconocimiento en el engine.

Esta API contiene muchos más objetos que la de alto nivel. A continuación, se detalle como es su funcionamiento:

1. La aplicación determina dónde debe ser enviado el audio y crea un objeto de audio destino a través del cual el engine envía sus datos. Microsoft ha implementado un objeto de audio destino utilizando el dispositivo multimedia wave-out, pero la aplicación puede utilizar destinos de audio personalizados como ficheros WAV.

2. La aplicación, a través del enumerador, localiza un engine de texto hablado que pueda utilizar. Entonces, crea la instancia del objeto engine y la pasa al objeto de audio destino.

3. El objeto engine tiene una caja de diálogo con los objetos de audio destino que puede utilizar para encontrar un formato de datos compatible para el audio digital. Una vez aceptado el formato, el engine crea una notificación que se pasa al objeto de audio destino. A partir de ahí, este objeto envía información al engine a través de la notificación.

4. La aplicación ya puede registrar la notificación principal que recibe notificaciones independientes del buffer.

Cuando un objeto de gramática reconoce el habla, envía la notificación a la cadena

5. Cuando todo está preparado, la aplicación pasa uno o más buffers de texto al engine. Estos buffers serán hablados por el engine automáticamente.

6. Para saber qué palabras están siendo habladas, la aplicación crea un buffer de notificaciones por cada objeto buffer. Cuando el engine habla una palabra, busca en un marcador y llama a las funciones del buffer de

notificación. La notificación es vaciada cuando el buffer ha terminado de enviar los datos para su reproducción.

MÓDULO DE DICTADO

La API Speech también contiene un módulo para crear dictado con las palabras y frases enviadas por el usuario. Actualmente, Speech está en la revisión 4.0 Beta, por eso, estos artículos estarán basados en la versión 3.0 de la API. El módulo de Dictado requiere la versión 3 o superior. Una aplicación puede saber si se ha instalado la versión 3.0 utilizando el método QueryInterface, el cual fallará en versiones inferiores de la API.

El módulo de Comandos de voz permite, como ya sabemos, que una aplicación utilice fácilmente el reconocimiento de voz para el "Command and Control". Las aplicaciones sólo tienen que proveer una lista de comandos que se puedan entender y serán notificados cuando sean hablados por el usuario.

Para el módulo de dictado, el funcionamiento es similar, y permite dictar en las aplicaciones. La infraestructura de este módulo y sus interfaces son similares a una edit box (caja de edición de texto). El objeto dictado utiliza de forma metafórica una edit box invisible. Cuando el usuario dice una palabra, el texto es introducido directamente dentro de esta caja de texto. Seguidamente, la aplicación es notificada de que el texto de la caja ha cambiado y cómo lo ha hecho. Puesto que la caja no puede ser vista, la aplicación necesita visualizar cualquier cambio al usuario. Por ello, una aplicación debería mantener su propia caja de edición de texto y sincronizarla con la caja invisible para mostrar el estado actual de los datos.

Estamos de acuerdo en que si una aplicación sólo quiere transcribir las palabras habladas por el usuario, sería más fácil pegar el resultado de la frase final al engine de dictado mediante una caja de texto. Lo malo, es que perderíamos muchas de las funcionalidades que nos permite este estupendo control. Por ejemplo, soporta formateo de texto automático (capitalización, espaciado), puntualización de palabras en símbolos, construcción de glosarios, comandos limitados, y un GUI para que los usuarios puedan corregir el engine de dictado. Además, tiene potencial suficiente para ser ampliado en el futuro, especialmente, cuando aparezca el dictado continuo.

REQUISITOS HARDWARE Y SOFTWARE

Una aplicación que soporte dictado requiere cierto hardware y software instalados en el PC para poder trabajar. No todos los ordenadores tienen la memoria, velocidad, micrófono y altavoces necesarios para soportar la API Speech, por lo tanto, hay que diseñar las aplicaciones para que sea opcional su utilización.

Speech contiene un módulo para crear dictado con las palabras

Los requisitos hardware y software deberían ser considerados cuando se diseña la aplicación para el Speech:

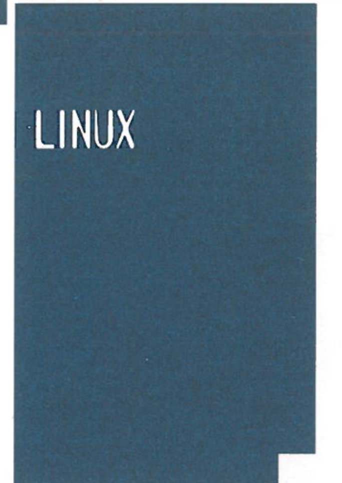
- Velocidad de procesador: el reconocimiento de voz y texto hablado requieren actualmente un Pentium 60 o superior. '
- Memoria: el reconocimiento de voz para el dictado consume de 4 a 8 megabytes de RAM, además de la requerida por la propia aplicación.
- Tarjeta de sonido: Cualquier tarjeta de sonido funcionará, incluyendo Sound Blaster, Media Vision, ESS Technology, las compatibles con Microsoft Windows Sound System, y el hardware de audio de ordenadores multimedia. Es posible que algunos engines de reconocimiento necesiten una tarjeta con DSP (Procesador de señales digitales).

El objeto dictado utiliza de forma metafórica una edit box invisible

- Micrófono: El usuario puede elegir entre 2 tipos de micrófonos: de sobremesa o de solapa (el cual está más cerca de la boca y siempre a la misma distancia). Para el dictado es mejor el segundo tipo, aunque se puede utilizar cualquier otro de mediana calidad.
- Sistema Operativo: Windows 95/98 o Windows NT versión 3.5.
- SPEECH: este engine debe estar instalado en el ordenador del usuario (Versión 3 o superior).

POR ESTE MES ES SUFICIENTE...

En este primer artículo nos hemos adentrado en lo que es la tecnología del reconocimiento de voz y todas sus posibilidades. En el siguiente número aprenderemos muchos más detalles sobre el módulo de dictado y requisitos hardware-software para un buen soporte de la API. Además, comenzaremos a desarrollar el primer programa.



Profundizando en SVGALIB (II)

Jorge F. Delgado Mendoza (ernar@convex.es)

¿Hasta dónde podemos llegar con SVGALIB?, la respuesta parece casi un slogan de Microsoft: hasta donde tú quieras. Desde el dibujo de formas simples hasta la superposición de pixmaps con doble buffering para crear un universo 3D, las únicas limitaciones las impone nuestra imaginación. En este segundo artículo continuaremos trabajando con esta librería gráfica.

En el anterior artículo comenzamos lo que ha de ser un pequeño curso de programación gráfica utilizando la excelente librería gráfica denominada SVGALIB. Esta herramienta nos permite acceder, de una manera cómoda y homogénea, al sistema gráfico de nuestro PC.

La razón fundamental para utilizar esta librería en lugar del servidor de *X-Windows* reside en la necesidad de tener un control más directo del *hardware* que tenemos a nuestra disposición. ¿El motivo? muy sencillo: velocidad. Al no incurrir en las sobrecargas impuestas por el protocolo X nuestras aplicaciones, - fundamentalmente juegos, - podrán aprovechar hasta el último ciclo de CPU de nuestro ordenador.

En el artículo anterior estudiamos un modo de instalación de SVGALIB en nuestro sistema y estudiamos la arquitectura de la misma. En el resto del artículo, describimos un buen número de funciones de la librería, encaminadas a la inicialización y a la obtención de información sobre las capacidades de nuestro sistema.

En esta entrega, vamos a profundizar un poco más en nuestro estudio de las facilidades que nos brinda esta importante herramienta. Comenzaremos el artículo por una revisión de los requisitos que debemos cumplir para sacar el máximo provecho de la información que se presenta.

Una vez establecidos los requisitos, estudiaremos dos puntos muy importantes a la hora de realizar programación gráfica: la organización interna de la memoria de vídeo y los diferentes sistemas de presentación de colores en pantalla.

Mientras revisamos ambos conceptos, aprovecharemos para introducir las primitivas gráficas que tienen relación con ellos, presentando los primeros ejemplos de código. Como viene siendo costumbre, todos los programas que se muestran en el artículo, así como otros ejemplos que el autor considera interesantes, se pueden encontrar en el CD que acompaña a la revista.

Concluido el apartado más teórico del artículo, pasaremos a describir las funciones de dibujo, las cuales nos van a permitir realizar todo tipo de figuras geométricas: líneas, segmentos, círculos, puntos, cajas, etc. A lo largo de todo el artículo haremos hincapié en el apartado práctico, presentando al lector distintos ejemplos en los que pueda ver el funcionamiento de las diferentes primitivas. Una vez analizadas las primitivas de dibujo, habrá llegado el momento de hacer una breve recopilación de todo lo que hemos aprendido, preparando al lector para la próxima entrega de la serie.

En el CD de la revista podremos encontrar una actualización de la librería gráfica, siendo su novedad más destacada el soporte de varias tarjetas de vídeo que

se encuentran con frecuencia en nuestros PC's :*S3 Trio64* y *S3D VIRGE*. Los pasos para poder instalarla son los mismos que se indicaron para la versión anterior, puesto que ella misma se encarga de borrar los restos de versiones que sean más antiguas. La versión que se encuentra en el CD es la 1.2.13, por lo que si tenemos instalada una copia más moderna en nuestro PC, es preferible no realizar ningún cambio. En cualquier caso, los ejemplos que se presentan han sido compilados con la versión 1.2.13.

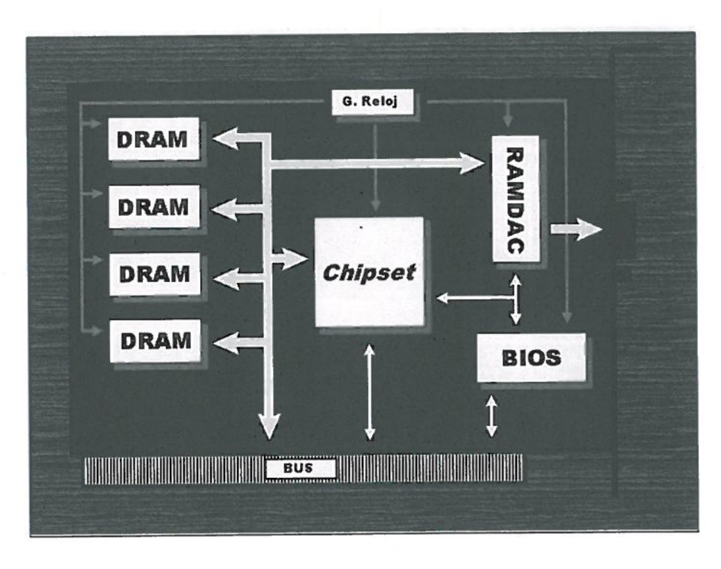


Figura 1. Arquitectura de una tarjeta de vídeo. En esta figura se muestran los diferentes componentes que constituyen una tarjeta gráfica.

REQUISITOS

- **Conocimientos:** la librería SVGALIB, como la mayoría de las aplicaciones existentes para *Linux*, está implementada en lenguaje de programación C. Esta es la razón por la que todos los programas de ejemplo incluidos en el artículo están escritos en este lenguaje.

  Por este motivo, es imprescindible una buena base de C para poder seguir las muestras que se presentan para ilustrar los aspectos más relevantes de la librería, así como para crear aplicaciones basadas en la misma.

  Con el fin de automatizar las tareas de compilación, vamos a utilizar el *GNU make*. Gracias a ella vamos a poder definir todos los parámetros de la creación de ejecutables mediante *Makefiles*, por lo que también conviene estar familiarizado con esta herramienta.

  En el apartado de aplicaciones, conviene conocer las peculiaridades del compilador - en nuestro caso **GCC**, - para ser capaces de agilizar al máximo el código. La diferencia de velocidad entre una compilación estándar y un ejecutable optimizado puede llegar a ser de varios órdenes de magnitud. Además, la utilización de determinadas opciones del compilador puede ser muy útil a la hora de depurar nuestros programas.

  También es importante conocer, aunque no sea a fondo, el subsistema gráfico de un PC. Dominar los conceptos básicos, - profundidad de color, resolución, funcionamiento de una paleta de colores, ordenación de los puntos en la memoria de vídeo, etc, - es muy importante para poder avanzar sin dificultad. Aunque el punto anterior es fundamental, a lo largo del artículo comentaremos por encima algunas de las ideas más relevantes, para intentar ayudar a que aquellos lectores que no estén familiarizados con los modos VGA.

- **Software:** necesitaremos tener instaladas las siguientes herramientas:

  - *gcc* 2.7.0 o superior.
  - *make*.
  - SVGALIB 1.2.10 o superior.

- **Conceptos Básicos:** lo primero que debemos hacer antes de comenzar a utilizar la librería, es aclarar algunos conceptos básicos relativos al sistema de vídeo VGA. Dado que los modos de vídeo con menos de 256 colores - 8bpp[1] - están en desuso y sólo se mantienen por compatibilidad con aplicaciones antiguas, vamos a ignorarlos en nuestra descripción del subsistema de vídeo, centrándonos en los modos de 8, 16, 24 y 32 bpp.

  Los dos conceptos que vamos a revisar son:

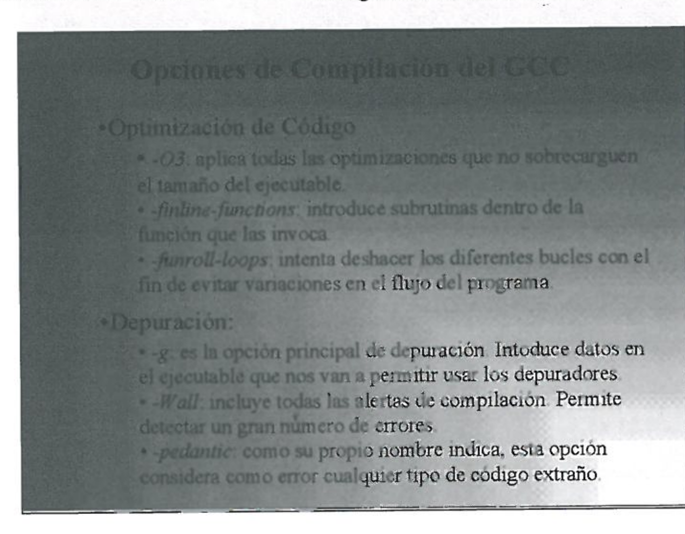


Figura 2. Esta tabla describe diferentes flags del compilador de GNU, los cuales se pueden utilizar a la hora de compilar nuestros programas. En la figura se incluyen opciones para optimizar el código, así como para facilitar la depuración del mismo.

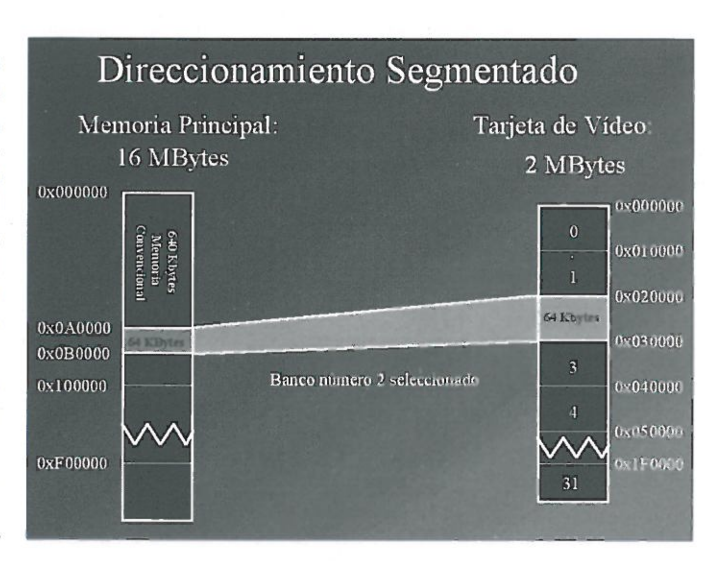


Figura 3. Acceso segmentado a la memoria de vídeo. Siempre podemos acceder a un bloque de 64 Kbytes en el rango de direcciones 0xA0000 - 0xB0000. Aunque la memoria de vídeo es un ente plano y contiguo, necesitamos rutinas especiales de cambio de banco para conmutar de un segmento a otro.

- El modo en el que la información se almacena en la memoria de vídeo, de tal manera que aparezca correctamente en nuestro monitor.
- La diferencia entre el significado de la información relativa a cada punto entre los modos de paleta, - con 256 colores, - y los modos de color *RGB*[2], directo o color verdadero - 32768, 65536 ó 16.7 millones de colores. –

ARQUITECTURA DE LA MEMORIA VÍDEO

La arquitectura de la memoria de vídeo es muy simple. Consiste en un *buffer* lineal de datos, que guarda la información correspondiente a cada punto. Para presentar la imagen en la pantalla, un componente de la tarjeta de vídeo, - el *RAMDAC*, - se encarga de convertir el número de *bytes* necesario para obtener un punto, en una señal analógica que el monitor sea capaz de representar.

El acceso a la memoria se realiza de forma secuencial, encontrándose generalmente el punto (0,0) de la pantalla en la dirección 0 de la memoria de vídeo. Si seguimos leyendo la memoria de una manera ordenada obtendremos a continuación los puntos (1,0) , (2,0), etc., hasta completar una línea. A continuación tendríamos el punto (0,1), (1,1), (2,1) y así sucesivamente.

Sin embargo, la BIOS del sistema sólo reserva 64 *Kbytes* en el mapa de memoria para gráficos *VGA*, limitando las resoluciones disponibles a 320x200x256. Para esquivar esta limitación, los fabricantes solucionaron dicho problema creando el concepto de *banco*. Una tarjeta permite acceder a toda su memoria en trozos de 64 *Kbytes*, mediante la utilización de unos registros especiales, posibilitando así la obtención de modos de resolución superiores.

La desventaja de este sistema es evidente. Por un lado, tenemos que crear rutinas de cambio de banco que son incompatibles entre las distintas tarjetas - de ahí la necesidad de *drivers* - y además hemos de estar continuamente comprobando si hemos llegado al final de un banco.

Para resolver esta situación, todas las tarjetas modernas implementan lo que se denomina un frame buffer lineal. Este sistema consiste en colocar el mapa de memoria de la tarjeta en un segmento libre del mapa de memoria del PC, de tal manera que podamos acceder a toda ella de una manera secuencial. Las figuras 3 y 4 muestran la diferencia entre ambas aproximaciones.

Las funciones básicas de SVGALIB acceden directamente a la memoria de vídeo, encargándose la librería de las operaciones de cambio de banco

¿En qué modo nos afecta a nosotros todo esto? Si nos limitamos a las funciones gráficas que ofrecen VGALIB y

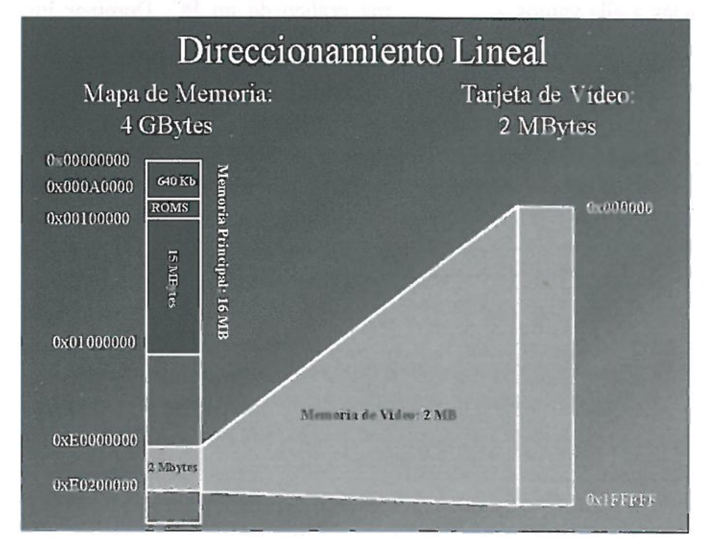


Figura 4. Acceso lineal a la memoria de vídeo. La mayoría de las tarjetas nuevas permiten recolocar toda su memoria en una posición muy elevada del mapa de memoria del PC. Así se evita la sobrecarga debida a las rutinas de cambio de banco, permitiendo un acceso más rápido.

VGAGL, en nada. Sin embargo, si queremos realizar operaciones algo más complejas como hacer copias de RAM a la memoria de vídeo, optimizar la velocidad de presentación, etc., tenemos que tener presente el funcionamiento interno del sistema.

Existen tres funciones que nos permiten manipular el sistema de bancos de una tarjeta de vídeo. Estas funciones, - útiles siempre que utilicemos modos de más de 64 Kbytes de memoria, - son las siguientes:

- vga_setpage(int #banco): esta función permite acceder al banco número #banco a través de 0xA0000-0xB0000. Los primeros 64 Kbytes corresponden al banco 0, los siguientes al 1, etc. En un modo de 1024x768x256 colores, tendríamos las primeras 64 líneas en el banco 0, las siguientes 64 en el 1, y así sucesivamente.

- vga_setwritepage(int #banco): esta función es similar a la anterior, pero sólo fija el banco que se accede cuando escribimos en 0xA0000-0xB0000. En la mayoría de las tarjetas de vídeo es posible definir un banco para lecturas y otro para escrituras, lo que permite reducir el número de cambios de banco en el caso de una operación de copia.

- vga_setreadpage(int #banco): de manera similar a la anterior, esta función permite definir el banco de memoria que vamos a acceder mediante una lectura a 0xA0000-0xB0000.

Como podemos ver, el sistema de cambio de bancos resulta bastante engorroso. Normalmente utilizaremos funciones de las librerías SVGALIB y VGAGL para dibujar, lo que nos evitará preocuparnos de seleccionar el trozo de memoria activo, ya que todas las ellas lo hacen automáticamente (pagando un cierto precio en el rendimiento final). A pesar de todo, este sistema no es muy eficiente.

Tabla 1. Ejemplos y funciones.

Ejemplo	Fichero	Funciones
1	ejemplo1.c	vga_init() vga_disabledriverreport() vga_setmode() vga_getmodeinfo() vga_setpalette() vga_setcolor() vga_drawpixel() vga_flip()
2	ejemplo2.c	vga_init() vga_setmode() vga_setpalette() vga_setcolor() vga_drawline()
3	ejemplo3.c	vga_init() vga_hasmode() vga_setmode() vga_setrgbcolor() vga_drawline() vga_drawscansegment() vga_getscansegment()
4	ejemplo4.c	vga_init() vga_disabledriverreport() vga_setpalette() vga_drawline() vga_setcolor() Copia directa en memoria de vídeo (Scr->Scr Blt)

Si la tarjeta permite utilizar direccionamiento lineal, el acceso a memoria es mucho más rápido y las funciones de VGAGL podrán ejecutarse en mucho menos tiempo. Para ver si es posible utilizar este modo de acceso, podemos utilizar la función vga_getmodeinfo(). En caso afirmativo, existe otra primitiva que permite colocar la tarjeta en modo lineal de operación. Se trata de la función:

- set_linearaddressing(): esta función coloca la tarjeta de vídeo en modo de direccionamiento lineal. La función devuelve -1 si el intento ha sido fallido, o bien el tamaño en bytes de la apertura lineal.

En el ejemplo número 1 repasamos los conceptos que hemos visto hasta ahora, incluyendo los que describimos en el primero de los artículos. Este programa se limita a presentar en pantalla información sobre diferentes parámetros de la tarjeta de vídeo, para después acceder a la memoria de vídeo realizando un pequeño dibujo.

El acceso a la memoria se hace de forma secuencial

En la Tabla 1 se muestran los diferentes ejemplos y las funciones que ilustran el artículo.

Todos los ficheros fuente, así como el Makefile, se encuentran en el CD-ROM de la revista.

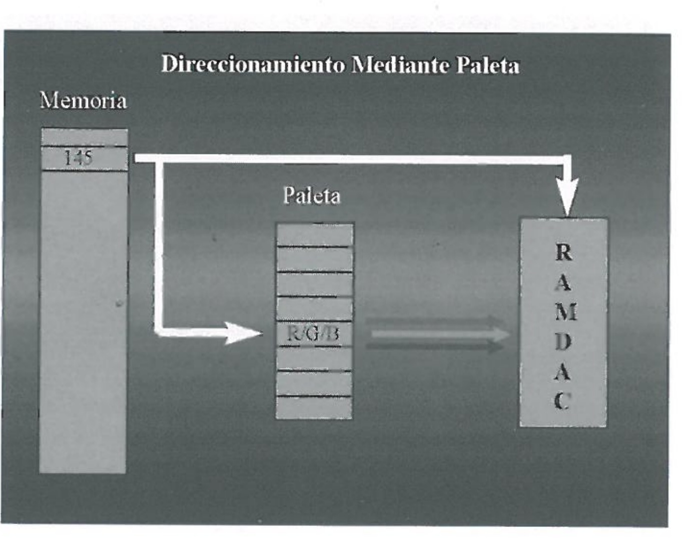


Figura 5. En el sistema de selección de color mediante paleta, los contenidos de la memoria no se envían directamente al RAMDAC de la tarjeta de vídeo. Los diferentes valores se emplean para direccionar una tabla que contiene los verdaderos valores de cada color.

PACKED PIXEL VS. PALETTE MODE

Para dibujar en la pantalla, hemos de trasladar a la memoria de vídeo del sistema la información relativa a las posiciones y colores de los diferentes puntos que componen nuestro dibujo. En el punto anterior, dedicado a la organización de la memoria, veíamos como hacerlo directamente a partir de la posición base de la memoria. Si utilizamos las primitivas de la librería, - tanto de SVGALIB como de VGAGL, - podemos olvidarnos de cómo está distribuida la memoria y limitarnos a proporcionar a las funciones coordenadas 'X' e 'Y' relativas a la esquina superior izquierda de la pantalla.

Sin embargo, a la hora de seleccionar el color, tenemos dos posibilidades:

- En los modos de 256 colores, se emplea una técnica denominada color de paleta o Palette Mode.
- En los modos de 15/16bpp y 24/32 bpp se emplea lo que se denomina color RGB o también Packed Pixel Mode.

Cuando se definió el estándar VGA, el máximo número de colores que se podían presentar en la pantalla eran 256, es decir 8 bits por cada punto. Como este número de colores permitía muy poco margen, se determinó que estos 256 colores pudieran elegirse dinámicamente de entre una tabla con 6 bits para cada uno de los componentes: Rojo, Azul y Verde. De esta manera, lo que representa el byte de información que se coloca en la pantalla de vídeo no es más que un índice a una tabla de 256 casillas, en cuya posición n-sima se guarda el valor tonal del color n-simo. La figura 5 muestra el modo de operación de este sistema, con el cual se conseguía una reproducción más fiel de las imágenes que con un sistema de 256 colores fijos.

Para manipular esta tabla, la librería gráfica SVGALIB nos ofrece las siguientes funciones:

- vga_setpalette (índice , r , g , b): esta nos permite actualizar los valores reales del color cuyo índice en la tabla utilizamos como primer argumento. Los valores r, g y b se refieren a las componentes RGB del color y pueden tomar valores entre 0 y 63.

Tabla 2. Ejemplo 2.C.

```
#include <stdio.h>
#include <vga.h>
#include <vgagl.h>


void
main()
{
        int i;
        /* Primero inicializamos la librerÌa */
        vga_init();
        /* colocamos un modo de 256 colores, este siempre existe */
        vga_setmode(G320x200x256);

        for ( i = 1 ; i < 64 ; i++ )
                {
                        /* colocamos en la posiciÛn i, un valor  [0-63],0,0 */
                        vga_setpalette ( i , i-1 , 0 , 0 );
                        /* ahora dibujamos el gradiente de lÌneas */
                        vga_setcolor ( i );
                        vga_drawline ( 0  , i+50 , 319 , i+50 );

                }

        while ( getchar() != ‘q’ )

        vga_setmode(TEXT);
}
```

- vga_getpalette (índice , *r , *g, *b): imagen especular de la función anterior, vga_getpalette() permite obtener los valores de color asociados a un índice concreto. Los componentes RGB se colocan en las direcciones a las que apuntan r, g y b.

- vga_setpalvec (inicio , cantidad , *paleta): esta función permite actualizar la información de color de cantidad número de puntos, a partir del índice inicio. Los datos relativos a los valores RGB de cada color se pasan en el array paleta, el cual contiene al menos cantidad de ternas RGB. En cada terna los valores pueden oscilar entre 0 y 63.

- vga_getpalvec (inicio , cantidad , *paleta): de nuevo la función inversa. Esta vez podemos obtener el contenido de color de las entradas que van desde inicio hasta inicio+cantidad en el vector de ternas apuntado por paleta.

El ejemplo número dos selecciona un modo de 256 colores y coloca todos los rojos puros en las posiciones 1 a 64 de la paleta de colores. Una vez concluida la tarea, presenta todos los colores así definidos en la pantalla. Como en todos los demás ejemplos, presionando la tecla q se finaliza la ejecución del programa.

Como ya es costumbre, todos los programas de prueba se pueden encontrar en el CD que acompaña la revista

En el sistema conocido como Color Directo, cada punto posee la información sobre los diferentes componentes RGB, con lo cual no es necesario utilizar una paleta. La desventaja es que cada punto debe ocupar más de un byte en la memoria de vídeo del PC. En el caso de los modos de color directo, no es necesario manipular la paleta, aunque los puntos ocuparán 2 bytes en los modos de 16 bits (65536 colores) y 3 ó 4 bytes en los de 16.7 millones de tonos (24 ó 32 bits).

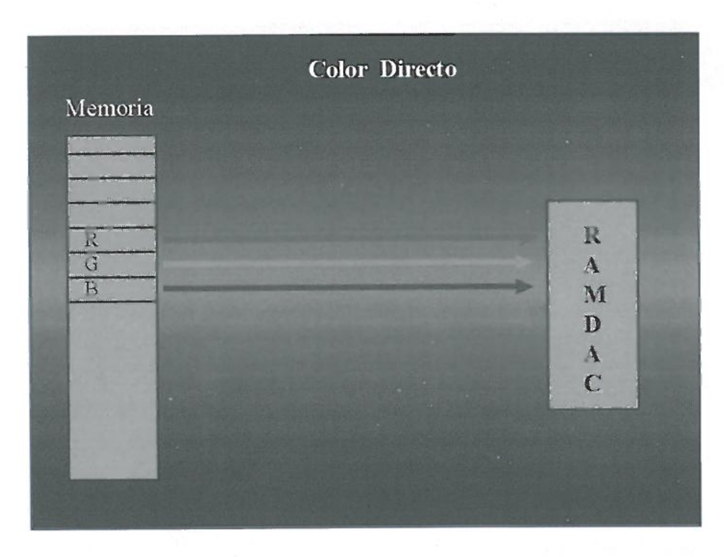


Figura 6. En los modos de acceso directo, la información guardada en la memoria de vídeo contiene los valores RGB de cada punto. De esta manera, utilizamos más de un byte para mantener toda la información necesaria.

DIBUJANDO CON SVGALIB

Una vez hayamos asimilado todos los conceptos explicados hasta ahora, ha llegado el momento de comenzar a dibujar formas simples. Para ello podemos utilizar las primitivas básicas de dibujo que están incluidas en nuestra librería. Estas funciones básicas acceden directamente a la memoria de vídeo, encargándose la librería de las operaciones de cambio de banco - en el caso de que fueran necesarias. Por motivos que veremos más adelante, no es aconsejable mezclar su uso con las rutinas de la librería VGAGL.

Estas funciones nos van a permitir crear líneas en direcciones arbitrarias, cambiar los colores y leer/escribir puntos. A continuación describimos cada una de las funciones y su utilización:

Función vga_clear()

Esta función no toma ningún argumento, ya que su utilidad es muy básica: borra la pantalla con el contenido del color cuyo índice es '0'. Esta entrada es, a no ser que se haya cambiado, el color negro. En el caso de los modos de color directo, el concepto de índice no existe. En estos casos se borra la pantalla con el color RGB (0,0,0).

Función vga_setcolor()

Esta función se utiliza para indicar el color activo de dibujo de la librería. El índice del color elegido se envía como parámetro de la función. En el siguiente ejemplo colocamos como color activo aquél cuyo índice en la paleta sea el 100.

```
vga_setcolor(100);
```

Todas las primitivas de dibujo que no posean información de color en sus parámetros utilizarán este color para sus actividades. Esta función sólo es relevante cuando tenemos modos de paleta, ya que el concepto de índice es irrelevante en los modos de color directo. Para estos últimos debemos utilizar la función: vga_setrgbcolor().

Función vga_setrgbcolor()

De manera similar a la función anterior, este primitiva se encarga de colocar un color determinado como activo. Esta función sólo tiene sentido para modos de 16, 24 y 32bpp. En este caso, como el

Tabla 3. Ejemplo 3.C.

```
#include <stdio.h>
#include <vga.h>
#include <vgagl.h>

void
main()
{
     int i;
     unsigned char colores[404];

     /* Primero inicializamos la librerÌa */
     vga_init();

     /* comprobamos si el modo existe */

     if ( vga_hasmode(G640x480x16M32) == 0 )
          {
               /* Nuestra tarjeta no permite el modo requeridó */
               printf ("El driver de la tarjeta no permite un modo\n");
               printf ("de 640x480 en 24 bits de color.\n\n");

               printf ("Imposible ejecutar el ejemplo.\n");
               vga_setmode(TEXT);
               exit (1);
          }
     vga_setmode(G640x480x16M32);

     /* borramos la pantalla */

     vga_clear();

     /* colocamos un color como activo.. es un verde claro */

     vga_setrgbcolor ( 10, 63, 10 );

     /* dibujamos un cuadrado usando lineas */

     vga_drawline ( 50 , 50 , 50 , 150 );
     vga_drawline ( 50 , 150 , 150 , 150 );
     vga_drawline ( 150 , 150 , 150 , 50 );
     vga_drawline ( 150 , 50 , 50 , 50 );

     /* y lo rellenamos usando scansegment... para ello hemos de inicializar
          el array de colores */

     for ( i = 0 ; i < 201 ; )
          {
               colores[i] = (i); /*R*/
               colores[400-i] = (i); /*R*/
               i++;
               colores[i] = 0; /*G*/
```

concepto de paleta no existe, será necesario indicar los valores de todos los componentes de la tonalidad. Cada uno de los factores RGB que forman el color puede oscilar entre 0 y 255. Para colocar un verde claro, podríamos utilizar la siguiente línea:

```
vga_setrgbcolor ( 0 , 255, 0 );
```

Función vga_drawpixel()

Esta función dibuja un punto en la pantalla. Sus coordenadas vienen determinadas por los parámetros que le pasemos. El color que se utiliza para el punto ha de ser definido mediante llamadas a vga_setcolor() o vga_setrgbcolor() en función del modo de vídeo que estemos utilizando.

Cuando se definió el estándar VGA, el máximo número de colores que se podían presentar simultáneamente en la pantalla eran 256

Para pintar un punto blanco en la posición (100,100) de nuestra pantalla, en un modo de 16 bits, podríamos utilizar la siguiente secuencia:

```
vga_setrgbcolor ( 255 , 255, 255 );
vga_drawpixel ( 100 , 100 );
```

Nunca, insisto nunca, debemos utilizar esta función - ni su homóloga, vga_getpixel(), - para dibujar más de un punto. La sobrecarga que introducen es enorme. Si queremos trasladar una imagen a la pantalla hay maneras mucho más rápidas, las cuales veremos más adelante.

Función vga_drawline()

De manera similar a la función anterior, esta primitiva dibuja una línea cuyas coordenadas de inicio y final se pasan como parámetros. El color ha de ser definido previamente con las funciones que ya conocemos. Esta función conviene utilizarla para cualquier tipo de línea, mientras no sea horizontal. El motivo es que el algoritmo que utiliza, llamado bresenham, resulta poco eficiente en ese caso. Un ejemplo que dibuja una línea desde el punto (0,0) hasta el (100,150) en color azul intenso, en un modo de 256 colores, sería el siguiente:

```
vga_setpalette ( 10 , 0 , 0 , 35 );
vga_setcolor ( 10 );
vga_drawline ( 0 , 0 , 100 , 150 );
```

Función vga_drawscansegment()

Esta función se utiliza para crear líneas horizontales. Su velocidad es muy superior a la de la función anteriormente descrita, ya que utiliza suposiciones acerca del modo de transferencia a través del bus del sistema - ver figura 1 - así como métodos especiales para conseguir accesos alineados a la memoria de vídeo. Dispone de cuatro argumentos que pasamos a detallar a continuación:

- El primero de ellos es un puntero a una cadena de tipo unsigned char. Como sabemos, cada uno de los elementos del array tendrá 8 bits. En un modo de 256 colores, cada uno de los bytes representará el color de un punto de la línea. En los modos de 16 bits, dos elementos consecutivos forman un color, mientras que en los modos de 24 y 32 bpp necesitaremos tres y cuatro elementos respectivamente, para tener la información de color de un punto.
- El segundo es la coordenada x de inicio de la línea.
- A continuación se da la coordenada y del comienzo del segmento.

Tabla 3. Ejemplo 3.C. (Continuación).

```
                colores[400-i] = 0;  /*G*/
                i++;
                colores[i] = i/4;  /*B*/
                colores[400-i] = i/4;  /*B*/
                i++;
                colores[i] = 0;  /*0*/
                colores[400-i] = 0;  /*0*/
                i++;
           }
    for ( i = 51 ; i < 150 ; i++)
           vga_drawscansegment ( colores , 51 , i , (99*4) );

    /* ponemos otra lìnea */

    vga_drawline ( 100 , 50 , 100 , 150 );

    /* ahora vamos a usar scansegment para copiar el cuadrado de un lado al otro */

    for ( i = 50 ; i <= 151 ; i++ )
           {
                vga_getscansegment ( &colores[0] , 50 , i , (101*4 ) );

                vga_drawscansegment ( colores, 400 , i , (101*4));
           }

    while ( getchar() != 'q' )

    vga_setmode(TEXT);

}
```

- Finalmente damos la longitud total del primer elemento. Este número representa la longitud de la línea que dibujar en bytes. En un modo de 256 colores será igual a la longitud de la línea, mientras que en los de 16, 24 y 32 el valor será el doble, triple y cuádruple, respectivamente de la longitud de ésta.

Función vga_drawscanline()

No es más que un caso particular de la anterior donde la longitud del segmento es toda la línea horizontal. El primero de los argumentos es idéntico al caso de vga_drawscansegment(), mientras que el segundo es la coordenada y de la línea.

Función vga_getscansegment()

Imagen especular de vga_drawscansegment(), esta función utiliza los mismos argumentos que la anterior. Sin embargo, su misión es guardar en el array de tipo unsigned char los valores que la memoria de vídeo tiene en las posiciones definidas por el resto de los parámetros de la función. En el ejemplo 3 se pueden observar demostraciones de todas estas funciones, en el cual se dibujan varios objetos en la pantalla utilizando las primitivas descritas.

La memoria de vídeo es un buffer lineal de datos

ACCESO DIRECTO A LA MEMORIA DE VÍDEO

A pesar de que las funciones antes descritas permiten realizar un gran número de tareas sobre la tarjeta de vídeo, hay veces en las que poder acceder directamente a la memoria significa ganar varios órdenes de magnitud en la velocidad del programa. Un caso típico sería el de trasladar una imagen que tenemos en la memoria principal del PC a la memoria de vídeo, con el fin de mostrarla por pantalla.

Con las primitivas que hemos estudiado, podríamos dibujarla utilizando vga_setrgbcolor() y vga_drawpixel(). Sin embargo este método es, por decirlo de una manera discreta, enormemente lento. Aunque otra manera más rápida de trabajar sería tener la imagen en un conjunto de cadenas de datos de color y utilizar vga_drawscansegment() con los parámetros apropiados. Dependiendo de la tarjeta, el modo de vídeo, etc., es posible conseguir ganancias de hasta mil veces utilizando este método.

La librería SVGALIB, como la mayoría de las aplicaciones existentes para Linux, está implementada en lenguaje de programación C

Finalmente, la manera más rápida de hacerlo sería mediante el acceso directo a la memoria de vídeo de la tarjeta, utilizando punteros a ésta y a la zona donde tengamos guardada la imagen. De esta manera minimizamos la sobrecarga de funciones y conseguimos toda la velocidad que puede dar la tarjeta[3]. En nuestro caso querríamos algo que nos permitiera realizar la siguiente tarea:

```
register unsigned char* origen;
register unsigned char* destino;
register int i;

for ( i = 0 ; i < tamaño de línea ; i++ )
{
    *destino++ = *origen++;
}
```

Para todas y cada una de las líneas de la imagen que tengamos. En principio esto es posible únicamente si conocemos la dirección de destino dentro del área de memoria de la tarjeta. Pues bien, pensada especialmente para este tipo de operaciones está la siguiente función:

Función vga_getgraphmem()

Esta función devuelve un puntero de tipo unsigned char a la posición de memoria inicial de la apertura, sea segmentada o lineal, de la memoria de vídeo. De esta manera, bastará con introducir la siguiente línea en el anterior pseudocódigo para que todo funcione a la perfección:

```
destino = vgagetgraphmem ();
```

La única pega de este método es que, en modos de vídeo de más de 64K, tenemos que encargarnos del cambio de banco de la tarjeta.

CONCLUSIONES

A lo largo de este artículo hemos aprendido a utilizar la librería SVGALIB para acceder a la tarjeta de vídeo. Hemos realizado ejemplos de obtención de información de la tarjeta, así como de utilización de las rutinas básicas de dibujo. Finalmente, hemos aprendido a acceder directamente a la memoria para así poder realizar lo que se denomina imageblt, o traslado de imágenes, de una manera óptima. Este método que hemos tratado es muy importante, ya que la mayoría de los juegos utilizan bitmaps y no primitivas básicas de formas geométricas.

MÁS ALLÁ: VGAGL

Las funciones de acceso a la memoria de vídeo que nos ofrece SVGALIB son útiles, pero se quedan cortas para una gran cantidad de aplicaciones. Al utilizar directamente la memoria de vídeo, no podemos realizar funciones tan típicas como el page flipping que evita que el usuario vea cómo se dibujan los diferentes objetos. Tampoco podemos usar double buffering para que el paso de una imagen a otra sea suave y continuo. En fin, que tenemos una gran cantidad de limitaciones que nos impone el bajo grado de abstracción de la librería.

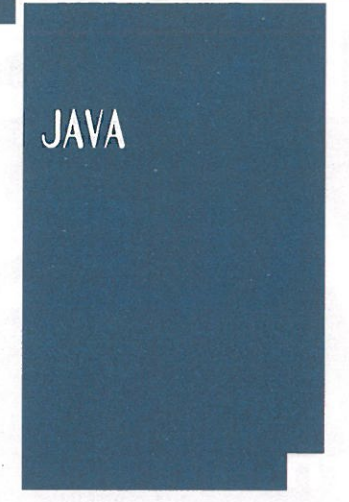
Para paliar esta situación, nació VGAGL. Esta librería permite definir pantallas virtuales en la memoria del sistema, contextos de dibujo, sistemas rápidos de traslado de imágenes, conmutación entre contextos y pantallas virtuales, etc. En el próximo artículo estudiaremos VGAGL en profundidad, describiendo las ventajas que ofrece.

---

1 bpp: acrónimo de las palabras inglesas bits per pixel. Indica en número de colores accesibles en un determinado modo de vídeo. 8 bpp indica que cada punto posee 8 bits con información de color, lo que nos permite seleccionar 256 colores diferentes. Valores típicos para los diferentes modos son 8, 16, 24 y 32 bpp.

2 RGB: acrónimo de Red Green and Blue, representando los colores básicos en función de los cuales se puede obtener cualquier otro tono.

3 Es posible conseguir aun más velocidad utilizando accesos directos a memoria con rutinas en ensamblador, pero un estudio detallado de este tipo de métodos está más allá de la intención de este artículo.



Utilización de threads (I)

Javier Sanz Alamillo (jsanz@a01-unix.uc3m.es)

La utilización de threads (literalmente hilos) permite la ejecución simultánea de varios flujos de control que pueden realizar diferentes tareas e incluso interactuar entre ellas. Mediante el uso de Java se pueden desarrollar programas con hilos de una forma simple y efectiva, permitiendo hacer uso de nuevas y potentes tecnologías.

Como usuarios estamos familiarizados con las aplicaciones *multithread*. Por ejemplo, los procesadores de texto imprimen los documentos sin que por ello uno se tenga que detener a que termine la impresión, se puede continuar escribiendo un nuevo texto o continuar el actual. Incluso, los navegadores de Internet visualizan varias imágenes a la vez, sin que ello nos pueda parecer nada extraño. Todo esto es posible gracias a la programación con *threads*.

¿QUÉ ES UN THREAD?

Un *thread* (hilo o contexto de ejecución) es un flujo de control de un programa, esto es, una secuencia de instrucciones en ejecución. El concepto que debemos acuñar como *thread* es similar al de proceso, excepto que múltiples *thread* pueden ejecutarse dentro del mismo proceso, compartiendo los datos y el código de programa. La mayoría de los programas se pueden considerar como de *thread* único, es decir, existe un único camino de ejecución, el programa tiene un inicio, se realizan una serie de cálculos y se finaliza. Los programas multihilo pueden tener varios flujos de control, varios threads que se desarrollan en diferentes partes del código ejecutándose "simultáneamente".

En un proceso típico con varios *threads*, ninguno, uno o varios hilos pueden están ejecutándose a la vez. Cuando se dice "a la vez", hay que tener en cuenta el número de procesadores que tengamos en el ordenador sobre el que se ejecuta el proceso. Una máquina con <*n*> procesadores puede ejecutar hasta <n> *threads* en paralelo. Si únicamente se dispone de un procesador, se podrán ejecutar todos los *thread* del proceso compartiendo el tiempo que se le ha adjudicado a cada proceso, de forma que aparentemente todos los *thread* se están ejecutando simultáneamente.

CARACTERÍSTICAS DE LOS HILOS

Como ya hemos comentado, el uso de hilos en los programas permite la ejecución simultánea de varias tareas dentro de un mismo programa, además una mejora en rendimiento del mismo. Todo esto es posible gracias a las características de los *threads*, que se describen a continuación:

- Los *threads* de un proceso comparten el código que se ejecuta, luego sólo hay una porción de código ocupando memoria, siendo además una ocupación mínima. Además el tra-

siego de *threads* que tendrá que hacer el sistema operativo será escaso, entendiendo como tal el cambio que deberá realizar el sistema para colocar un proceso en ejecución y colocar el que se está ejecutando en espera. Este proceso se denomina cambio de contexto.

- Cada *thread* tiene su propia pila y sus propios valores para sus variables, por lo que se define su estado de ejecución, así como el flujo de control.
- Permiten un sencillo manejo de la memoria compartida, tanto en lo referente a datos como a variables, así como una forma práctica de comunicación entre ellos.

El uso de threads en un programa permite la ejecución "simultánea" de diferentes tareas

Las principales ventajas del uso de hilos frente a los procesos son que el cambio de contexto entre dos hilos es mucho más "barato" que entre dos procesos, ya que por ejemplo, el código de los diferentes *threads* como se ha comentado anteriormente es compartido. Además los hilos únicamente se diferencian entre sí por los valores de la pila y variable de cada hilo, y el tratamiento de memoria compartida resulta más sencillo y práctico.

Pero también la programación con *threads* tiene sus inconvenientes, siendo el principal debido a que durante el desarrollo de aplicaciones, resulta complicado el proceso de depuración. Esto es debido a que existen varios flujos de control que no se reproducen en el mismo orden debido a la asignación de tiempos (no olvidemos que aunque depuramos un código de programa único estamos tratando con varios flujos, varias ejecuciones simultáneas), y las herramientas que actualmente están disponibles no ofrecen muchas facilidades. Por lo tanto, programación con *threads* requiere cierto manejo, experiencia y paciencia, que como no, se ve recompensada por los resultados finales.

UN EJEMPLO PRÁCTICO

Un clásico de la programación tradicional es el conocido *HelloWorld*, como programa que visualiza ese mismo mensaje. Ese programa a su vez puede ser ejecutado mediante un *thread* como se detalla a continuación:

```
public class mensaje extends Thread {
  public void run() {
        System.out.println ( " Hello World
");
  }
  public static void main ( String args[] ) {
        mensaje miHilo = new mensaje ();
        miHilo.start();
  }
}
```

CREACIÓN DE THREAD

Para crear un nuevo *thread* es necesario generar una instancia de la clase *java.lang.Thread*. Un objeto de este tipo representa un verdadero hilo en ejecución para el intérprete de Java y es utilizado para el control y sincronización durante su aplicación. Mediante este objeto, podemos dormir el hilo, arrancarlo, suspenderlo, etc. Cabría esperar que el constructor de la clase *java.lang.Thread* sólo necesitase saber por dónde debería comenzar el hilo a ejecutarse, pero desafortunadamente no es así, por lo que para crear un *thread* se presentan dos alternativas posibles que vamos a exponer a continuación:

- Declarar el hilo con una clase extendida de la clase *java.lang.Thread*, mediante la herencia.
- Implementar la interfaz *java.lang.-Runnable*.

IMPLEMENTACIÓN FRENTE A HERENCIA

La implementación de la interfaz *java.lang.Runnable* es la forma habitual de crear *threads*, ya que proporciona una forma de abstracción a la hora de definir una clase. Pero antes de continuar con la implementación de un *thread* debemos recordar unas cuántas diferencias entre una interfaz y una clase.

Una interfaz sólo puede contener métodos abstractos, además de variables estáticas y finales. Las clases, por otra parte, tienen la posibilidad de implementar métodos y contener variables que no sean constantes.

Una interfaz no puede implementar cualquier método. Una clase que implemente una interfaz debe implementar todos los métodos definidos en esa interfaz. Además una interfaz tiene la capacidad de heredar de otras interfaces y al contrario que las clases, puede heredar de múltiples de ellas.

Para finalizar, una interfaz no puede ser instancia del operador *new*.

Recordando esto, un objeto que desee utilizar los métodos que proporciona la clase *java.lang.Thread* se puede declarar mediante la implementación de la interfaz *java.lang.Runnable*, que simplemente se define de la siguiente manera descrita:

```
public interface Runnable {
  abstract public void run();
}
```

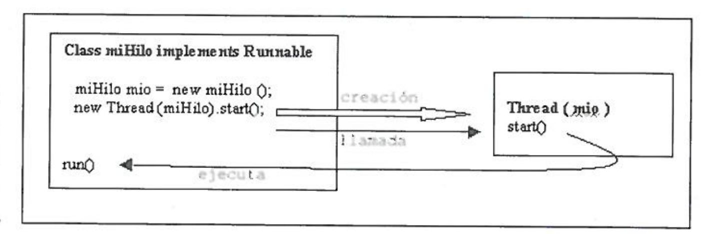


Figura 1. Creación de un thread mediante el método de implementación.

Con esto, el inicio de cada hilo quedaría definido mediante la ejecución del método *run*() para cada objeto creado, que contendrá el código que se va a ejecutar. Se debe destacar que *run()* es un método publico, no acepta parámetros y no lanza ninguna excepción. Por tanto, cualquier clase que contenga un método *run()* con las características anteriores, simplemente debe implementar la interfaz *java.lang.Runnable*. Cada objeto de esta clase sería *Runnable* y por tanto se podrá utilizar con la clase *java.lang.Thread*.

Una vez mostrados los fundamentos, pasemos a la parte práctica en la que se aprecia la sencillez de este método para generar hilos. Como se ha descrito, la clase debe implementar la interfaz *Runnable*, por lo que para empezar, nuestra clase debe tener la siguiente definición:

```
public class miThread implements Runnable {
}
```

Bien, nuestro hilo empieza a ejecutarse en el método *run()* de la interfaz que estamos implementando, por lo que añadiremos a la clase:

```
public class miThread implements Runnable {
        public void run () {
                // Código del hilo que ejecutar
una vez creado
        }
}
```

A partir de este momento ya hemos implementado la interfaz y el método, luego ya solamente queda poder utilizar nuestra clase como un verdadero objeto tipo de tipo *Thread*. Por lo tanto, generaremos un objeto tipo *java.lang.Thread* y le pasaremos un objeto del tipo de nuestra clase, de la forma mostrada en el ejemplo:

```
miThread miHilo = new miThread ();
new Thread( miHilo ).start();
```

Es posible indicar que comience la ejecución de nuestro hilo mediante el uso de los métodos que proporciona la clase *java.lang.Thread*, ya que hemos generado un objeto de esta última clase pasando como argumento nuestro hilo a ejecutar.

Como se observa el método que se utiliza es start(). Cabe esperar que fuera *run*(), pues es el método que tiene nuestro código a ejecutar, pero no es así, el método *start*() es el encargado de llamar a nuestro método *run*(). Esto se debe a que el método *start*() de *java.lang.Thread()* arranca el objeto tipo *Thread* como tal, y no como un método, lo que implica a continuación la llamada al método *run*() de nuestra clase. En la figura 1 se puede observar gráficamente este proceso.

HERENCIA

El otro método para la creación de hilos consiste en heredar un objeto de la clase *java.lang.Thread*. Se trata de una forma más sencilla que la anterior, ya que la propia clase *java.lang.Thread* contiene el método *run*() que ejecuta nuestro código. Por tanto, para que nuestra clase pueda ser ejecutada como un hilo, heredamos de la clase *java.lang.Thread* y escribimos nuestro código dentro del procedimiento *run*(), que tiene las mismas características que el método *run*() de la interfaz *Runnable*. Siguiendo los pasos descritos, seremos capaces de generar un hilo heredando de la clase *java.lang.Thread*, tendremos:

```
        class miHilo extends Thread {
}
```

Como la clase *java.lang.Thread* consta del método *run*() entonces sólo tenemos que redefinir nuestro método *run()* :

```
class miHilo extends Thread {
        public void run() {
                // Código a ejecutar
        }
}
```

La creación de nuestro hilo y su ejecución sería de la forma siguiente:

```
miHilo mio = new miHilo();     mio.start();
```

Como se observa, la ejecución también se realiza con la llamada al método *start()* tal y como se explicó en apartados anteriores.

La implementación de los *threads* mediante este método es muy interesante pero, por ejemplo, si se quiere crear un thread que herede de otra clase que no sea *java.lang.Thread*, como por ejemplo, *Applet*, no se permite. Ésto restringe considerablemente el empleo su empleo.

De los dos métodos detallados en el artículo, conceptualmente heredar de *java.lang.Thread* parece la mejor solución porque el objeto ya contiene el método *run*() y ya es de por sí (por la herencia) de la clase *java.lang.Thread*. Por tanto, se suele heredar de la clase *java.lang.Thread* cuando es posible (si nuestra clase no hereda de otras, debido a que Java no permite la herencia múltiple). Por otra parte, utilizaremos la interfaz *Runnable* cuando nuestra clase herede de otras, de tal forma que es el método más comúnmente utilizado.

ESTADOS DE UN THREAD

Durante el ciclo de vida de un *thread*, éste se puede encontrar en diferentes estados, los cuales pueden apreciarse en la figura siguiente. En ella aparecen dichos

estados y los métodos que provocan el paso de uno a otro. Este diagrama no es una máquina de estados finita, pero es lo que más se aproxima al funcionamiento de un *thread*, que como se aprecia es muy similar al proceso clásico.

Antes de pasar a detallar los estados de un *thread* cabe destacar que los *threads* son objetos que tienen como atributo predefinido su estado de ejecución. Éste no es directamente accesible como es habitual en un atributo de cualquier objeto, sino que su estado cambia a través de los métodos disponibles por la clase *java.lang.Thread*.

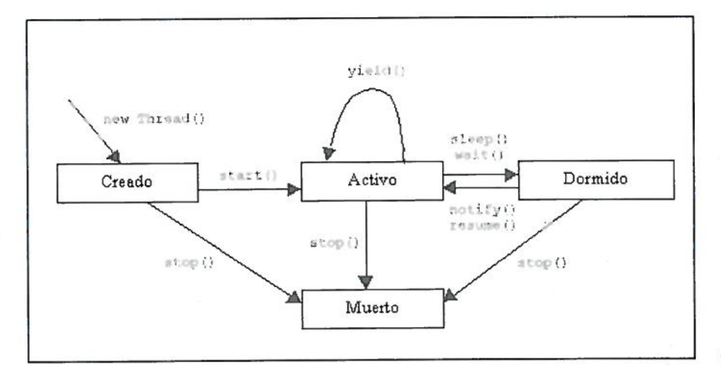


Figura 2. Estados de un thread.

CREACIÓN

La creación del *thread* se produce mediante los procedimientos anteriormente descritos, pero como se ha podido observar, su creación no implica de manera automática su ejecución, su arranque, sino que el *thread* pasa al estado de creado. Por ejemplo, suponiendo que creamos una clase *miClaseThread* que hereda de *java.lang.Thread*, tendríamos:

```
MiClaseThread miHilo = new miClaseThread();
```

También podríamos haber hecho:

```
Thread miHilo = new miClaseThread();
```

(Se entiende que *miClaseThread* hereda de *java.lang.Thread* y como éste es un objeto también de tipo *java.lang.Thread* la sentencia es correcta).

Cuando un *thread* se encuentra en este estado es simplemente un objeto de tipo *java.lang.Thread*, al que no se le asignan más recursos que la memoria necesaria. El *thread* esta aún sin activar, y sólo se puede arrancar usando el método *start*() o deternerlo definitivamente mediante el método *stop*(). En esta situación llamar a cualquier otro método carece de sentido y lo único que se provocaría sería el lanzamiento de una excepción de tipo *IllegalThreadStateException*.

ACTIVO O EJECUTABLE

Con la llamada al método *start*() se crearán los recursos necesarios para la que el *thread* pase al estado de activación, se incorporará a la lista de procesos disponibles para ser ejecutados por el sistema y se llamará al método *run*() de nuestro *thread*. En este momento el *thread* estará activo. Siguiendo la creación del ejemplo anterior, tendríamos:

```
miHilo.start();
```

Se denomina en estado activo y no "en ejecución" porque no se puede decir que el *thread* se esté ejecutando permanentemente en nuestro ordenador, es decir, que su estado sea activo no quiere decir que sé este ejecutando, aunque sí se encuentra en disposición de hacerlo.

En aquellos ordenadores con un único procesador no se pueden ejecutar *n threads* a la vez, por lo que para simular esta situación Java implementa un tipo de lista de prioridades que permite que el procesador sea compartido entre todos los *threads* que se encuentran en la lista. Gracias a la rapidez del procesador aparentemente para nosotros es como si se estuvieran ejecutando todos a la vez. Por tanto, un *thread* activo es aquel al que el procesador asigna regularmente un lapso o perodo de tiempo.

Cuando el *thread* se encuentre en este estado y se le asigne una porción de tiempo de procesador, todas las instrucciones de código que estén dentro del procedimiento *run*() serán ejecutadas.

DORMIDO

Un *thread* se encuentra en este estado si se llama al método *suspend*(), cuando se invoca el método *sleep*(), si el thread esta bloqueado por la espera de unos datos de la entrada/salida o cuando se realiza una llamada al método *wait*() para esperar a que se cumpla una determinada condición. En cualquiera de estas circunstancias el *thread* no consume recursos de la máquina.

Los métodos de recuperación o vuelta al estado activo dependerán, como es razonable, de la forma en que haya llegado al estado de dormido, siendo cada uno de ellos una forma exclusiva, por lo que tendremos:

- Si el *thread* está dormido por un periodo de tiempo, volverá al estado de activo al finalizarse dicho período.
- Si un *thread* esta suspendido mediante *suspend()*, se activa si se realiza la llamada al método *resume* ().
- Si el *thread* esta bloqueado por una entrada/salida, cuando ésta se resuelva, automáticamente se volverá al estado activo.
- Si el *thread* espera por una condición (invocación *wait()*), cada vez que la variable que controla la condición varíe debe llamarse a *notify*() o *notifyAll*().

Por ejemplo, un *thread* puede estar esperando una entrada/salida porque se requiera una entrada por teclado, con una sentencia del estilo:

```
System.in.read();
```

En nuestro procedimiento *run*() dormiría el hilo hasta que el usuario introdujera un dato por el teclado.

También podemos provocar la espera de un *thread* mediante el uso del método *sleep*(), por ejemplo:

```
class mensaje extends Thread {
        public void run () {
        System.out.println ("Me voy a dor-
mir");
        try {
                sleep ( 20*1000 );
                //tiempo en milisegundos
        }
        catch ( Exception e ) {
        System.out.println ("Que sueño
mas raro ...");
                }
        }
}
```

La línea de código *sleep* (20*1000) provoca que el hilo "duerma" durante veinte segundos. Durante ese tiempo transcurrido, aunque el procesador estuviera sin utilizar el *thread* no se ejecutaría. Tras esos segundos volverá al estado activo y entonces el procesador si le adjudicaría un lapso de tiempo cuando llegase su turno.

Este método tiene muchas aplicaciones, por ejemplo, si se quiere que el *thread* se active según una frecuencia de tiempo, o se lo que se desea es que el resto de hilos tengan más tiempo de ejecución, etc.

Cabe otra posibilidad es que el *thread* pase a dormido mediante *suspend*(), por lo que estaría en este estado hasta que no se ejecutara el método *resume*(). Por ejemplo:

```
class mensaje extends Thread {
 public void run ()        {
        while ( true ){
                System.out.println ( "hola
hola");
}
        }
public static void main ( String args[] )  {
mensaje miHilo = new mensaje();
miHilo.start();
System.out.println (" Veo muchos hola hola
... ");
System.out.println (" No me acaba de gustar
");
System.out.println (" Lo siento, los
paro...");
miHilo.suspend();
System.out.println (" Que tranquilidad ...");
System.out.println (" Bueno, no están tan
mal ...");
miHilo.resume();
}
}
```

El programa mezclará los mensajes **"hola hola"** del *thread*, junto con el resto de mensajes, pero cuando se ejecute el método *suspend*() sólo aparecerá **"Que tranquilidad ..."** sin mezclarse con los del *thread*. Al llamarse al método *resume()* el *thread* vuelve a activo y continuará visualizando infinitos **"hola hola"**.

MUERTO

El último estado en que puede estar un *thread* es el de muerto. A él se puede llegar de dos formas diferentes tal y como detallamos:

- Debido a una ejecución normal o debido al empleo del método *run()*.
- Por la ejecución del método *stop()*.

En el primer caso, el de la ejecución normal, la finalización proviene debido a que la tarea que se quería realizar ha finalizado,y se ha llegado al final del método run(). Por ejemplo:

```
public void run() {
        int veces ;
        for ( veces = 0 ; veces < 10 ;
veces ++ ){
                System.out.println (" Turno
" + veces );
        }
}
```

Cuando se haya visualizado el número **9** el método *run()* finaliza, por lo que el hilo también termina.

También se puede finalizar un *thread* llamadon al método *stop()*.

Si en el ejemplo de la clase mensajes se eliminan las llamadas a *suspend()* y *resume()*, y se cambian por un *stop*(), quedaría de la forma siguiente:

```
System.out.println (" Lo siento, los
paro...");
// Eliminamos miHilo.suspend();
System.out.println (" Que tranquilidad ...");
System.out.println (" Bueno, no están tan
mal ...");
// Eliminamos miHilo.resume();
miHilo.stop();
```

Al ejecutarse el método *stop*() se finalizaría el hilo, y no aparecerían más mensajes **"hola hola"** por pantalla. El hilo ha concluido de forma definitiva. Este método se encarga de enviar un objeto de tipo *ThreadDeath* al hilo que quiere detener, por lo que el *thread* morirá cuando reciba esta excepción.

Aparte de todos estos métodos descritos, existe un método que posibilita conocer en qué estado se encuentra un *thread*. Se trata del método *isAlive()*. Devuelve un valor lógico para diferenciar si el proceso se encuentra activo o dormido, por lo que obtenemos verdadero o falso si el proceso no ha sido creado o si está muerto.

Una vez que conocemos los estados de un *thread* y los métodos asociados, vamos a desarrollar un ejemplo en que se pueda apreciar la importancia de la programación con *threads*.

EJEMPLO PRÁCTICO

En el ejemplo vamos a simular una carrera entre una liebre, una tortuga y un caracol, de forma que cada uno será un objeto

de una clase "animal" que tendrá la velocidad de cada animal y su nombre. Cada objeto de la clase *Animal* será un *thread* que dormirá en función de su velocidad, de tal forma que, por ejemplo, el *thread* asociado a la liebre dormirá menos que el del caracol.

```
class fabula {   static Animal liebre ;
        static Animal tortuga;
        static Animal caracol ;

        public static void main( String args[] )
        throws  InterruptedException    {
                liebre  = new Animal ( 5, "L" );
                tortuga = new Animal ( 3, "T" );
                caracol = new Animal ( 1, "C" );

                liebre.start();
                tortuga.start();
                caracol.start();

                //El método join() consigue que
        hasta que no han
                //finalizado todos hilos la carrera no
        se acabe
                liebre.join();
                tortuga.join();
                caracol.join();
        }
}class Animal extends Thread{
        String nombre ;
        int velocidad ;

        public Animal ( int velocidad, String nombre
        ){
                this.nombre = nombre;
                this.velocidad = velocidad ;
        }

        public void run (){
                for ( int i = 0 ; i < 10 ; i ++ )  {
                        System.out.print ( nombre
        );

                        try     {
                                sleep ( 1000 / velo-
        cidad );
                }

                catch ( Exception e )   {
                        /* Nada */         ;
                }
        }
        System.out.println ( " \t " + nombre + "
                llegó " );
                }
}
```

Obtendríamos algo similar a:

```
LTCLTLLTLTCLLTLLTLTC   L llegó
TTTC  T llegó
CCCCCC          C llegó
```

Como se observa, la L (liebre) al ser más rápida llega antes que los demás a la meta, tras ella la tortuga y finalmente el caracol. Con este ejemplo se ha comprobado la creación de varios hilos mediante el método de la herencia, su ejecución, el estado de dormido mediante el método *sleep*() y la muerte natural tras finalizar el método *run*(). Es un buen ejercicio realizar este mismo programa sin programación con *threads*.

PRIORIDADES DE LOS THREADS

Java garantiza de cierta manera la gestión de los *threads* en cuanto al nivel de prioridad asignado a cada uno de ellos, de tal forma que cada hilo tiene una prioridad asociada. Si en un momento determinado un *thread* con prioridad mayor que el *thread* actual pasa a estado de activo, éste pasa a ejecutarse y el actual será dormido. Dicho de otra forma, mientras el *thread* de mayor prioridad no esté dormido, es él el que contará con los recursos. Si se duerme, se pasa a los otros hilos la posibilidad de utilizar los recursos, pero si vuelve a pasar a activo el *thread* de mayor prioridad, éste recupera de nuevo los recursos. Por defecto los *threads* con idéntica prioridad se ejecutan con el sistema del *round robin*, el cual define que si un *thread* comienza a ejecutarse, lo seguirá haciendo hasta que ocurra una de las siguientes circunstancias:

- Se llame al método *sleep*() o *wait*().
- Se espere una entrada/salida.
- Si se ejecuta un método sincronizado.
- Se llame al método *yield*().
- Se invoque el método *stop()*.

Gráficamente este método podría describirse tal y como muestra el esquema de la figura 3.

De todos modos, Java deja ciertos aspectos del cambio de *thread* a la implementación que estemos utilizando. El principal problema de este asunto es que no todas las implementaciones de Java usan el mismo sistema, ya que por ejemplo el entorno Windows utiliza la técnica de *time slicing* (fragmento de tiempo) para los *threads* con la misma prioridad. Con este método, el tiempo de procesamiento de cada *thread* viene determinado de forma que cada hilo se ejecuta durante un período de tiempo hasta que llega el cambio de contexto, esto es, se le termina el tiempo de procesador, y se ejecuta otro hilo, aunque se siguen manteniendo las mismas reglas para los thread con mayor prioridad. Debido a que Java no garantiza la fragmentación de tiempo, todas las aplicaciones que se diseñen deben tratarse como si se fueran a ejecutar con el sistema de *round robin*.

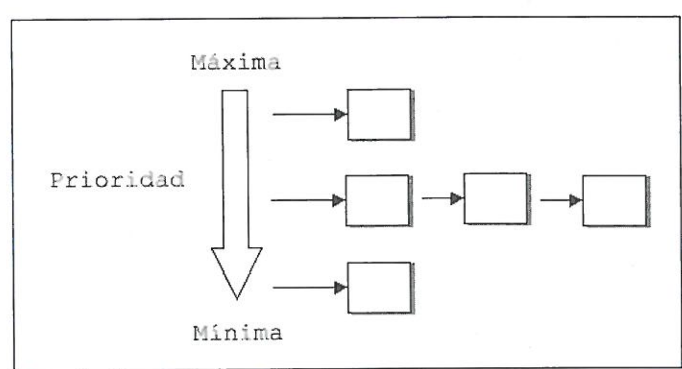


Figura 3. Método round robin.

TObject

Exception TStream TPersistent TComObject

TGraphicObject TGraphic TComponent TCollection TStrings

TApplication TDataSet TMenu TControl TCommonDialog TField

TGraphicControl TWinControl

Figura 1. Jerarquía de clases VLC.

a partir de la clase abstracta *TCustomEdit*. La forma de efectuarlo pasaría por establecer esas propiedades como *Protected*.

2) Desarrollar un componente visual, que heredará de *TWinControl*. Estos objetos poseen la propiedad Handle y se hacen visuales en tiempo de ejecución. Como ejemplo claro de todos estos componentes que heredan de *TWinControl* podemos fijarnos en los botones, cajas de texto, etc.

3) Controles gráficos, son todos aquellos controles que no necesitan interactuar con el usuario, es decir, no reciben nunca el foco de la aplicación. Como ejemplo más claro tenemos los *TLabel*, las etiquetas de texto que mostramos por pantalla sólo a modo descriptivo o informativo. Cuando queramos desarro llar componentes de este tipo, es recomendable heredar de la clase *TGraphicControl*.

4) Además, Windows permite crear nuestras propias clases, y por supuesto, C++ Builder también. Es el caso de componentes que ya tengamos diseñados en otros entornos y que basta con agruparlos en una DLL, y usarlos sencillamente en nuestro programa. Son las llamadas subclases.

5) Componentes no visuales. Son todos aquellos que no nos ofrecen una salida por pantalla. Como ejemplo fundamental tenemos los componentes *TTimer* y *TApplication*. Casi todos estos componentes heredan de *TComponent*, que será la clase referencia que usar a la hora de diseñar estos componentes.

PARTES DE UN COMPONENTE

Aunque las partes de un componente son las habituales en la programación tanto en C++ Builder como en el resto de herramientas de desarrollo visual, las recordaremos brevemente. Un componente consta fundamentalmente de tres partes que son:

1) Propiedades.
2) Métodos.
3) Eventos.

Las propiedades permiten leer y escribir valores en variables, para su posterior uso, son visuales en tiempo de diseño, permitiendo al programador cambiar sus valores iniciales, asignándoles el correcto en cada caso. Además, permite verificar la entrada correcta de datos restringiendo el posible valor de estas variables (p.e., sólo *True o False*).

Los métodos son bloques de código asociados a un determinado objeto, en este caso, el componente, y que al ser invocados realizan una serie de operaciones que el desarrollador del componente ha establecido. Por ejemplo, una llamada a un método *Open* podría ocuparse de la apertura de un fichero. De esta manera se ahorra al programador tareas bastante tediosas, puesto que un componente ofrece una serie de funcionalidades, encapsulando los detalles de la implementación. Es decir, simplemente tendremos que saber que métodos contiene cada componente y utilizarlos, sin necesidad de conocer la forma en que se realizan las acciones asociadas a ellos de forma interna.

Los eventos responden siempre a acciones o actividades desarrolladas en tiempo de ejecución. Asociando un determinado bloque de código a un evento podemos lanzar una determinada respuesta a dicho evento *(Event Handler)*, p.e., un *clic* de ratón.

Por tanto, el objetivo de un programador a la hora de crear un componen te consistirá en reducir el máximo trabajo posible al desarrollador de aplicaciones. De nada nos serviría crear un componente cualquiera y que el programador

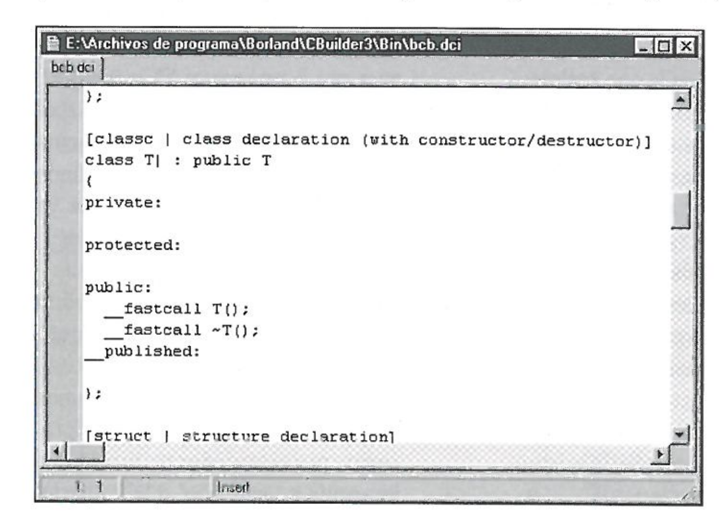


Figura 2. Clase de un componente.

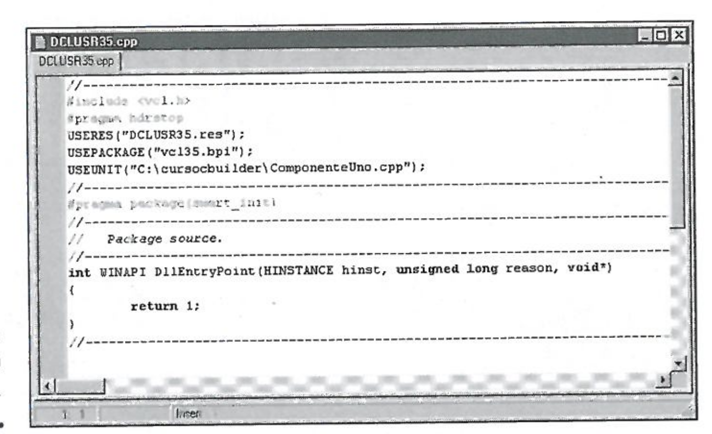


Figura 3.
Registro de un componente.

que lo vaya a utilizar tuviera que crear, por ejemplo, en el caso de componentes visuales, la ventana o asignarle el *Handle* a ese componente. Estas acciones las debe realizar el desarrollador del componente, bien heredando de otros componentes existentes, o simplemente escribiendo el código correspondiente.

REGISTRO DEL COMPONENTE

Antes de poder usar un componente es fundamental que lo registremos, es decir, que indiquemos a C++ Builder en qué paleta debe situar ese componente. Además podemos incluirlo en un registro de componentes, con lo que sólo instalando éste podremos instalar de golpe todos los componentes que hayamos desarrollado. Todo esto lo veremos un poco más adelante.

CREANDO UN NUEVO COMPONENTE

Existen dos formas posibles de crear un componente:

1) Usando el *wizard* o asistente de creación de componentes de C++ Builder.

2) Crear los componentes de forma manual.

Lógicamente, la primera opción es la más simple, aunque sea cual sea la elección siempre hay una serie de pasos que seguir, realizándolos de forma manual o asistida. Como se puede deducir de lo leído hasta ahora, la creación de un componente es como la creación de un módulo más para una aplicación:

1) Debemos crear una unidad donde vamos a incluir todo el código del componente.

2) Dicho componente debe heredar básicamente de alguna de las clases de la VCL, aunque sea solamente de la clase *TObject*, de tal modo que logre proporcionar ciertas funciones básicas de Windows (recordemos, *Handle* en caso de necesitalo, propiedad *Name*, asignación de memoria, etc.).

3) Añadiremos nuestras propiedades, eventos y métodos, incluyendo el código necesario para cada una de ellos.

4) Registraremos nuestro componente para poder usarlo en C-Builder, e incluirlo después dentro de un paquete (package) para poder usarlo en el IDE de Borland, o bien incluirlo en una DLL.

USO DEL ASISTENTE PARA LA CREACIÓN DE COMPONENTES

El asistente simplifica la creación de nuevos componentes, pero a costa de permitirnos un menor control sobre el proceso. Una de las mayores limitaciones del uso del asistente es la imposibilidad de añadir más componentes a la misma unidad CPP, y su fichero de cabecera asociado. Veamos en un ejemplo la creación de un componente. Lógicamente, lo primero es saber qué componente vamos a crear, elegir de dónde va a heredar y en qué paleta pretendemos situarlo. En este caso será un componente genérico que heredará de Tcom-

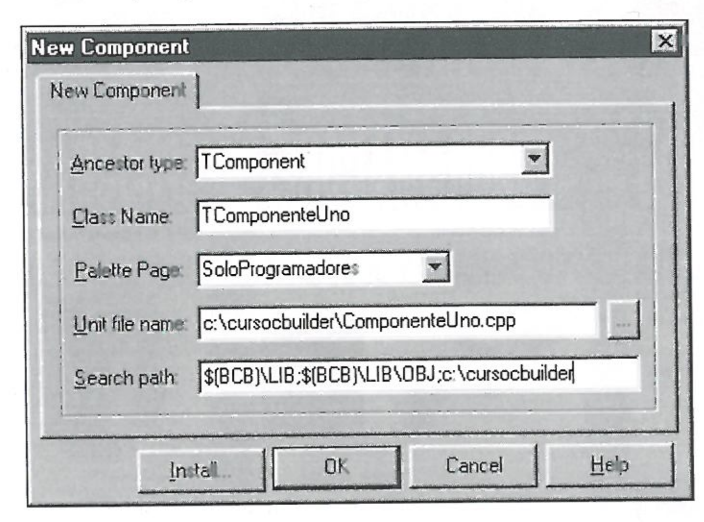


Figura 4.
Asistente de creación de componentes.

números atrasados

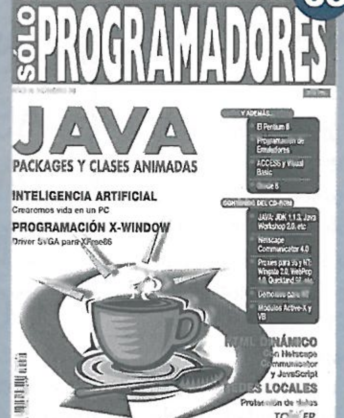
SÓLO PROGRAMADORES

completa ya tu colección

35



36



37

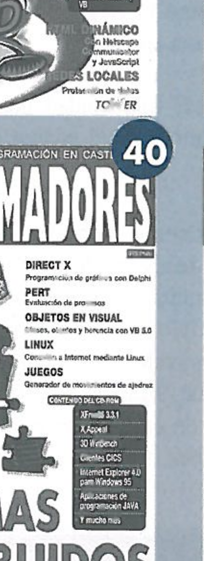
38



39

¿QUÉ ES EL UML? ANÁLISIS Y DISEÑO OO

40



41



GESTIÓN DE MEMORIA DINÁMICA CON C

42



MODELADO DE OBJETOS

43

INTELIGENCIA ARTIFICIAL
COMPUTACIÓN EVOLUTIVA





44

LA CONEXIÓN CON INTERNET
PROGRAMACIÓN DE SOCKETS





45

ATENDER PETICIONES DE PÁGINAS HTML
CREACIÓN DE UN SERVIDOR WEB CON DELPHI





46

UNA OPCIÓN DE GESTIÓN POTENTE Y VELOZ





☑ **Sí, deseo suscribirme a la revista SÓLO PROGRAMADORES durante un año (12 números) eligiendo la siguiente modalidad.** (Marcar con una X)

☐ Suscripción **NORMAL** 9.350 ptas.

☐ Suscripción **ESTUDIANTES** 7.050 ptas.

Nombre .... Apellidos ....
Domicilio: calle .... Nº .... Piso ....
C.P: .... Población: .... Provincia: ....
Teléfono: .... Fax: .... E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº............... .... Fecha de caducidad................ / ....
(VISA ELECTRÓN NO VÁLIDA)
☐ Domiciliación bancaria. (Rellenar código cuenta cliente)
☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

CODIGO DE CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

•ESTA OFERTA ANULA LAS ANTERIORES
•Oferta válida hasta fin de existencias •Válido para Peninsula y Baleares

SÓLO PROGRAMADORES

☑ **Sí, deseo recibir los siguientes números atrasados** ....
....
.... **Importe total** .... **ptas.**
de SÓLO PROGRAMADORES al precio unitario de 1.250 ptas. (IVA incluido)

números 1, 2, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 23, 24 y 26 agotados

Nombre .... Apellidos ....
Domicilio: calle .... Nº .... Piso ....
C.P: .... Población: .... Provincia: ....
Teléfono: .... Fax: .... E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº............... .... Fecha de caducidad................ / ....
(VISA ELECTRÓN NO VÁLIDA)
☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

•ESTA OFERTA ANULA LAS ANTERIORES
•Oferta válida hasta fin de existencias •Válido para Peninsula y Baleares

PEGAR AQUÍ

Rte.: ____________________

____

____

RESPUESTA COMERCIAL
Autorización nº 14.107
B.O. de C. nº 36
del 18.04.97

NO
NECESITA
SELLO
(A franquear
en destino)

HOJA DE PEDIDO



APARTADO FD Nº 214
28080 MADRID

suscríbete

a Sólo Programadores y consigue un magnífico descuento

suscripción **normal**

ahorro **20%**

12 revistas (1 año) por sólo...

9.350 ptas.

suscripción **estudiantes** (carreras técnicas)

ahorro **40%**

12 revistas (1 año) por sólo...

7.050 ptas.









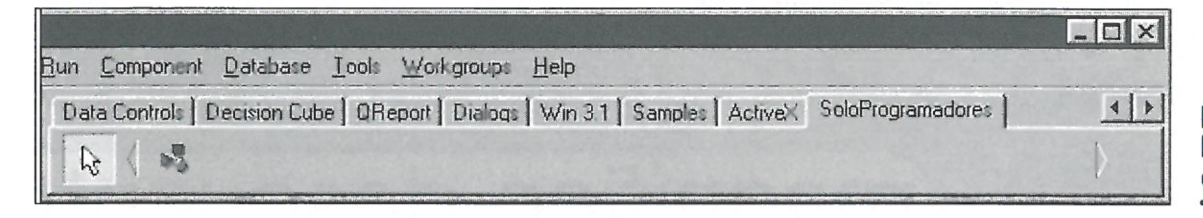


Figura 6. Paleta de componentes.

ponent y cuyo nombre será el de **ComponenteUno.**

En primer lugar deberemos lanzar el asistente, para lo que seleccionaremos la opción **New Component** del menú Components. También tenemos la posibilidad de acceder a la creación de un nuevo componente a través de la opción **New** del menú **File**, y en la plantilla seleccionar **New Component.**

Rellenamos ahora los valores de la clase antecesora, el nombre de nuestra nueva clase, la paleta donde vamos a dejar nuestro componente y cómo se va a llamar. Puede usar los valores de la imagen o usar otros cualquiera.

A continuación se detallan los valores incluidos en cada uno de los campos de la figura 4.

1) En el apartado *Ancestor Type* hemos seleccionado la clase de la cual vamos a heredar, y que ofrecerá una serie de eventos, propiedades y métodos.

2) En el apartado *Class Name* hemos escrito el nombre de nuestro nue vo componente, es decir, cómo vamos a llamar a la clase que se va a crear.

3) Posteriormente hemos introducido una paleta que llamamos **Sólo Programadores**, donde se colocarán todos los componentes desarrollados en los artículos de esta serie.

4) En *Unit File name*, introduciremos el nombre y el *path* en donde guardaremos la nueva clase.

5) Y en caso de haber elegido un directorio que no está en el *Search Path*, debemos añadirlo en esta última pestaña.

De esta forma hemos creado nuestro primer componente, que debido a su sencillez no ofrece grandes resultados. Lógicamente, al heredar desde *TComponent* poco podíamos esperar que hubiese aportado C++ Builder. Antes de nada grabe a disco la nueva unidad CPP creada con el el nombre por defecto en el directorio donde se creó el componente.

Sólo nos queda instalarlo en un determinado *Package* para poder usarlo directamente en el IDE de C++ Builder. Para ello seleccione la opción **Install** del menú Component, y seleccione su nuevo componente.

Seleccione en la opción **Unit File Name** el fichero *CPP* que ha creado anteriormente. Por ahora, no vamos a meternos en la creación de un nuevo paquete, simplemente use el que C++ Builder propone por defecto. Pulsaremos el botón, apareciendo un aviso que nos pide recompilar la paleta de componentes. Pulse **Ok** de nuevo y espere unos momentos mientras se compila y se enlaza el paquete con el nuevo componente. Una vez finalizado, el entorno avisa que ha añadido un nuevo componente a la paleta.

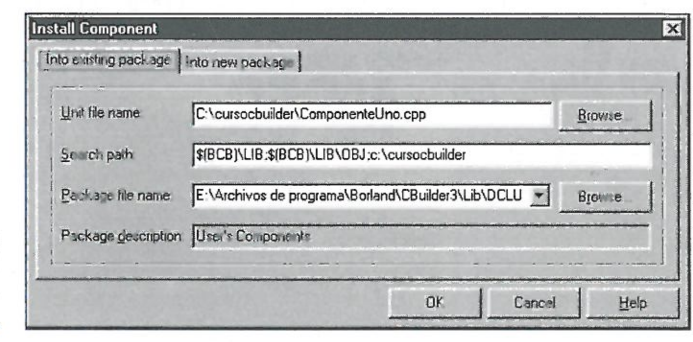


Figura 5. Selección del package.

Compruebe el componente creado en la pestaña *Sólo Programadores*. Realmente su funcionalidad es mínima, permitiendo que lo coloquemos en un formulario de la misma forma que con cualquier otro. No está mal para no haber escrito ni una sola línea de código.

Si el lector es curioso, y espero que lo sea, ya habrá echado un ojo al fichero *CPP* creado y al fichero de cabecera. En próximas entregas, al hablar de la creación manual de componentes, explicaremos ampliamente todos estos aspectos.

DE VUELTA CON LOS PAQUETES

Como ya se ha referido anteriormente todos los componentes de C++ Builder son instalados y almacenados en paquetes. Lo normal es que todos los componentes que creemos, y que estén interrelacionados entre ellos, se encuentren asociados en un mismo paquete. La creación de un paquete requiere:

1) Un nombre para el paquete.

2) Una lista de los paquetes requeridos por nuestros componentes.

3) Una lista de unidades que añadir, que contendrán los componentes

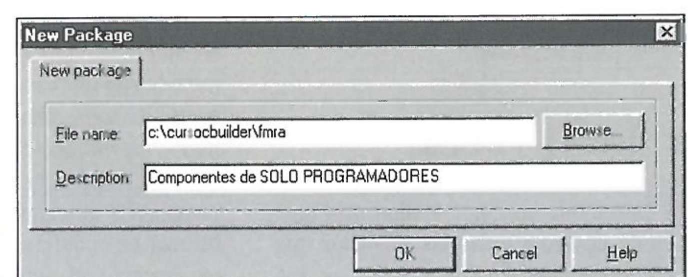


Figura 7. Creación de un nuevo paquete.

propiamente dichos, o cualquier unidad necesaria.

Vamos a crear un paquete denominado FMRA, en el que se incluirán todos los componentes de este curso. El método para crear los paquetes es muy simple y lo detallamos a continuación.

En pimer lugar cerraremos todo lo que estuviese abierto en el IDE de C++ Builder. Seleccionamos la opción **New** del menú **File** y pulsamos sobre el icono **Package**. Grabe este paquete con un nombre, por ejemplo **FMRA**.

A continuación veremos que contiene y qué necesita este paquete, para lo que seleccionaremos la opción **Project Manager** del menú **View** del menú principal. Como podemos observar aparecen dos carpetas, una de elementos requeridos y otra de elementos contenidos. En estos últimos actualmente sólo tenemos la propia unidad que engloba al paquete, pero en cuanto a los elementos requeridos aparecen todas las unidades necesarias de la VCL de Borland.

Vamos a añadir nuestro componente al paquete. Para ello seleccionemos **Project** en el menú principal y a continuación la opción **Add To Project**. Seleccionaremos nuestro componente desde el directorio donde lo hayamos creado, y pulsaremos **Ok**.

Si volvemos al *Project Manager*, vemos que hemos añadido a la carpeta *Contains* una nueva entrada denominada **ComponenteUno.cpp**, es decir, nuestro componente.

Vayamos un poco más adelante seleccionando la opción **Options** del menú **Project**, con lo que aparecen una serie de opciones distribuidas por pestañas. Por ahora, fijémonos sólo en la pestaña *Description*, donde existen tres botones de opción en *(RadioButtons)*. Mediante ellos podemos decidir si nuestro paquete va a ser usado sólo en tiempo de diseño, sólo en tiempo de ejecución o en tiempo de diseño y en tiempo de ejecución.

Esta última opción es la que nos interesa en este momento, por lo que la seleccionaremos. En este momento hemos llegado al final del proceso y solamente falta grabar todo, efectuar un **Build** de nuestro proyecto e instalar nuestro paquete en el IDE de desarrollo de Borland.

Antes tendremos que quitar el componente que antes habíamos añadido, para lo que seleccionaremos la opción **Component** del menú principal y posteriormente **Install Package**. Pulsa sobre el *check* del paquete *User Components* y selecciona la opción **Remove**. Pulsa **Ok** y el componente anterior desaparecerá.

Ahora repetiremos la misma operación pero para añadir nuestro nuevo paquete, para lo que tendremos que pulsar en **Add** y luego buscar el fichero **fmra.bpl**, que contiene el paquete en cuestión. Automáticamente aparece el componente, la paleta de componentes *Sólo Programadores* y el paquete en la lista de paquetes.

CONTROL DE ACCESO. UN POCO DE CLASES Y OBJETOS

Fundamentalmente vamos a referirnos al hecho de que existen cinco niveles de acceso para propiedades, métodos y variables de cada uno de los objetos. Cada uno de esos niveles de acceso es el encargado de determinar la disponibilidad o no de cada una de esas partes de la clase, los cuales vamos a describir a continuación de forma detenida:

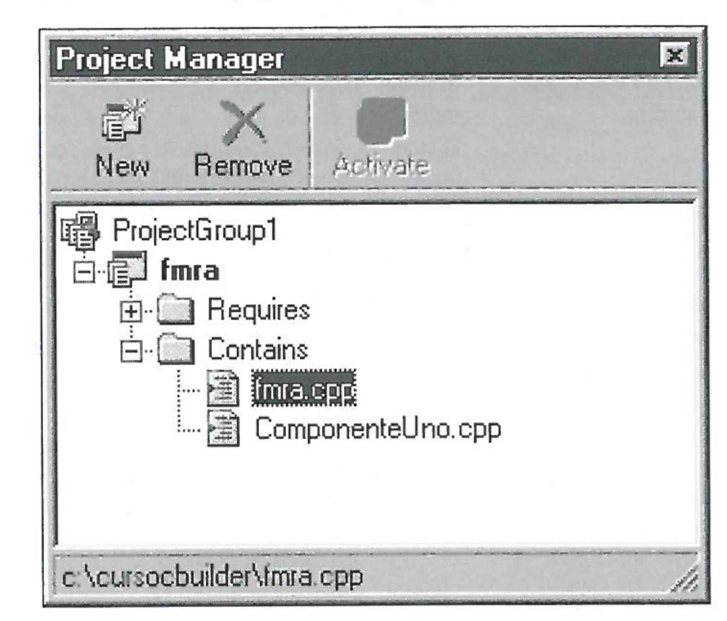


Figura 8. Project Manager.

1) *private*: El acceso está permitido solo y exclusivamente al código de la unidad de desarrollo del componente. Es el nivel más restrictivo de todos.

2) *protected*: El acceso a estas partes de la clase está permitido a todo el código de la clase y a todos los objetos que descienden de la misma.

3) *public*: el acceso está permitido desde todas las partes del código, desde donde se llamen.

4) *_automated*: Es similar al anterior, pero sólo para objetos de tipo *OLE*.

5) *_published*: Es el nivel menos restrictivo de todos y permite el acceso desde cualquier parte del código y desde el Inspector de Objetos. En esta parte declararemos las propiedades que deseemos modificar en tiempo de diseño.

PARTE PRIVATE

Declarar parte de una clase dentro del apartado *private* hace invisible el código ajeno a esta clase, por lo que será muy útil para almacenar partes del código que no deseemos que sean usadas por los desarrolladores de aplicaciones, permitiéndonos modificarlas sin que el desarrollador de aplicaciones necesite modificar ni una línea de su código. Es el nivel de acceso por defecto, usado si no determinamos por código otro diferente.

PARTE PROTECTED

Cuando deseemos que ciertas partes del código de nuestro componente sean accesibles por futuros componentes que hereden de él, debemos referir estas porciones de nuestra clase en la parte *protected* del código fuente de la clase.

De este modo, aunque desde una unidad cualquiera no podamos acceder directamente a esas partes del código, sí que podemos crear otra clase que herede de la misma y acceder al código de esa nueva clase heredada.

PARTE PUBLIC

De este modo hacemos visible esta porción de código a todas las unidades que usen esta clase. El único 'pero' es que son visibles en tiempo de ejecución. Es muy útil declarar las variables que sólo van a poder usarse en tiempo de ejecución, y no mediante el Inspector de Objetos, en la parte pública de una clase, con el fin de evitar errores y problemas de fallos de inicialización.

PARTE _PUBLISHED

Cuando deseemos que una serie de propiedades puedan editarse desde el Inspector de Objetos, además de desde el propio código, tendremos que declararlas en la parte *_published* de nuestra clase. Lógicamente, además de las propiedades los eventos deberán estar en esta parte de la clase. Por el contrario, aquellas propiedades que sean sólo de lectura *(Read-Only)* no es aconsejable que se encuentren en la parte *_published* de clase.

DISPATCHING

Muchas veces nos encontramos con que métodos definidos en la clase 'padre', de la cual hemos heredado nuestra clase son insuficientes para las necesidades que tenemos. En este punto entra en juego el *dispatching*, o cómo una aplicación es capaz de reconocer qué debe usar, si lo definido para la clase padre o lo definido por nosotros en la clase heredada.

De este modo se definen:

1) Métodos regulares.
2) Métodos virtuales.

Los métodos regulares son todos excepto aquellos que determinemos nosotros como virtuales, es decir, es considerado como el tipo por defecto. Un método regular es heredado tal como esa través de todas sus clases 'hijas' Si definimos un método con el mismo nombre en la clase hija que en la clase padre, el compilador determina que aquel que debe usar es el definido en la clase hija, reemplazando el anterior. Pero para ello debe tener el mismo nombre y los mismos parámetros. De este modo es el compilador, en tiempo de compilación, quien determina qué se debe usar.

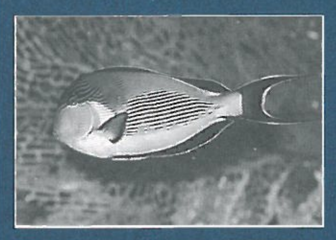
Public permite el acceso desde cualquier parte del código en el que tenga lugar la llamada

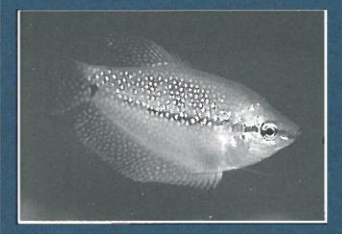
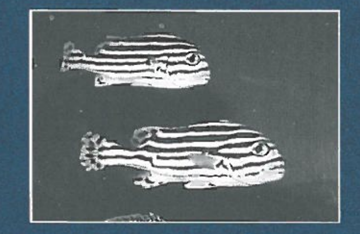
Por el contrario, los métodos virtuales son determinados en tiempo de ejecución. El compilador crea una tabla de métodos virtuales, cada una con los parámetros que se necesitan para invocar a esos métodos, y en tiempo de ejecución reconoce uno u otro, a través de esa tabla, usando el oportuno.

Un complemento a los métodos virtuales son los métodos *override*. Más que cambiar la definición completa del método padre, o usar la del hijo, complementa a la del padre añadiendo aquello que necesitemos en la clase derivada, pero siempre sin sobreescribir, sino complementando.

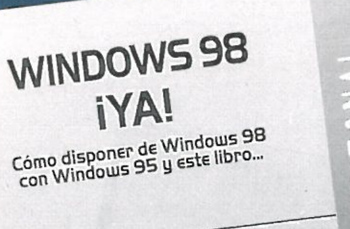
En el próximo número profundizaremos en la creación de componentes, añadiendo métodos, eventos y propiedades a un componente genérico, y adentrándonos en el difícil pero fascinante mundo de la construcción manual de componentes sin el uso de asistentes. De esta forma podremos aplicar de una forma más práctica los conocimientos adquiridos sobre la creación de componentes.

PARA  MOVERTE  EN

 INFORMÁTICA  COMO

 PEZ  EN 

EL  AGUA 









Incluyen CD-ROM de regalo



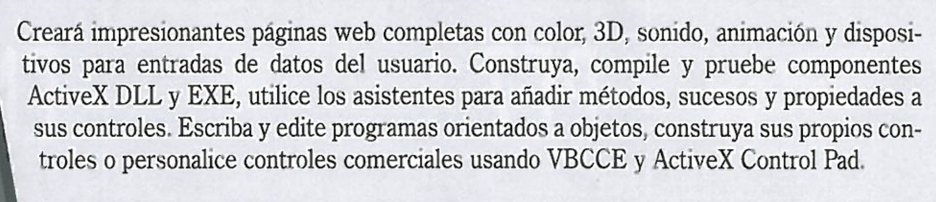
c/ Aragoneses, 7 • 28108 Alcobendas (Madrid)
Tel.: (91) 661 42 11* • Fax: (91) 661 43 86

Ya a la venta en quioscos y librerías.

LIBROS

Manual fundamental de ActiveX

Con este manual descubrirá cómo animar con ActiveX sus páginas web y otros programas basados en Windows. Aprenderá a construir, personalizar, probar, depurar y pulir los controles ActiveX usando instrucciones paso a paso y apoyándose en ejemplos reales. Desde la integración de controles estándar en páginas web hasta la creación de componentes dibujados por el usuario y de estilo documento.



Creará impresionantes páginas web completas con color, 3D, sonido, animación y dispositivos para entradas de datos del usuario. Construya, compile y pruebe componentes ActiveX DLL y EXE, utilice los asistentes para añadir métodos, sucesos y propiedades a sus controles. Escriba y edite programas orientados a objetos, construya sus propios controles o personalice controles comerciales usando VBCCE y ActiveX Control Pad.

El libro está acompañado por un CD-ROM para complementar sus contenidos, el cual contiene Visual Basic 5.0 Control Creation Edition, ActiveX Control Pad, Vivo Action Producer, etc.

Editorial: Anaya Multimedia
Nº páginas: 512
Nivel: Intermedio-avanzado
Autor: Richard Mansfield
Idioma: Español
Precio: 3.750 Ptas.(I.V.A. inc.)

Año 2000. Cómo sobrevivir a la crisis de la informática

Miles de compañías han salido ya de la etapa de negación y se han puesto en marcha para encontrar una solución. La mala noticia es que tenemos una emergencia informática mundial que necesitamos afrontar ahora. El 1 de enero de 2000 es una fecha límite que no podemos aplazar.



La buena noticia es que puede hacer algo con respecto a ésto. Existe el conocimiento técnico y tiene todas las herramientas disponibles hoy.

Con este libro sus autores pretenden ayudarle a desarrollar una estrategia para prepararse y protegerse de las ramificaciones de envergadura del 2000. Abrazando la perspectiva de que la situación es un desafío para la dirección más que un reto técnico, esta valiosa guía cubre a fondo los temas más importantes del control de gastos y le proporciona métodos prácticos para crear soluciones eficaces.

También le ofrecerá las ideas y los consejos prácticos que necesitará para hacer algo desde ahora mismo.

Editorial: Anaya Multimedia
Nº páginas: 224
Nivel: Intermedio-avanzado
Autor: Peter de Jager y Richard Bergeon
Idioma: Español
Precio: 1.995 Ptas.(I.V.A. inc.)

LIBROS

Windows 98 ¡YA!

¿Cómo disponer de Windows 98 antes de que realmente se comercialice? La respuesta no es sencilla, ya que aparentemente Microsoft ha estado trabajando en muchos puntos del núcleo del sistema, y un poco en la interfaz del mismo, pero por lo que se ha podido comprobar en las betas oficiales, los cambios efectivos no son tantos como se presumían en un principio.

En este libro se da una solución, con su Windows 95 y los elementos que adjunta el CD-ROM que acompaña al libro podrá disponer prácticamente de todas las funciones del nuevo Windows 98.

Gracias a todos los nuevos elementos que Microsoft ha ido publicando en estos últimos tiempos, por una parte para solventar los problemas de Windows 95, y por otra para mejorar las prestaciones y el rendimiento del sistema, ahora es posible construirse un Windows a medida con muchas de las opciones que se nos ofrecen en esta nueva versión.

Ponga Windows 95 en forma con la ayuda de esta obra y disfrute de un renovado sistema operativo más potente sin cambiar de ordenador.

Editorial: Abeto Editorial
Nº páginas: 192
Nivel: Intermedio-avanzado

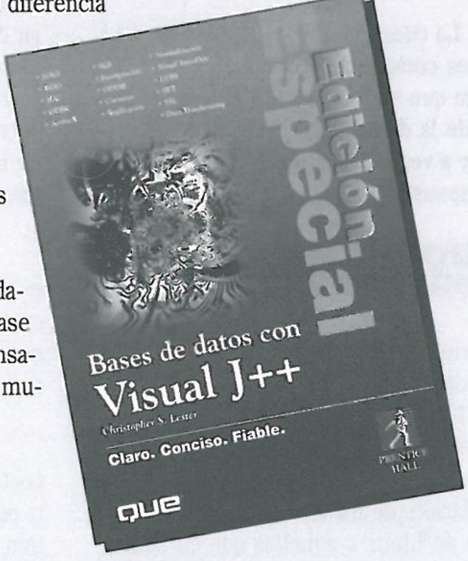
Autor: Javier Guerrero
Idioma: Español
Precio: 1.995 Ptas.(I.V.A. inc.)

Bases de datos con Visual J++

Este libro le va a permitir utilizar de lleno el poder de Visual J++ y aprender a construir una base de datos dinámica e interactiva dentro de la web. También llegará a aprender a conocer la diferencia entre una base de datos relacional y otra orientada a objetos y una multimedia con Visual J++.

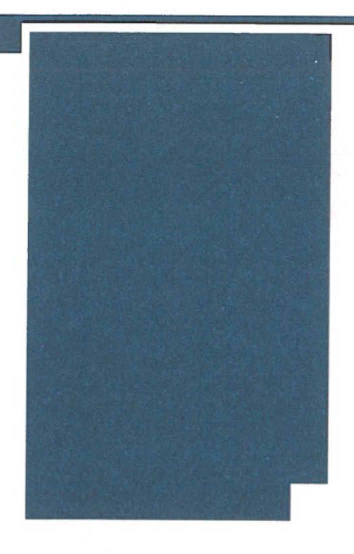
Cómo explorar los conceptos más avanzados de bases de datos, incluido el diseño, el modelado de datos, normalización e integridad referencial. Cómo trabajar con bloques estáticos, dinámicos, utilización del teclado, y todas esas nociones que le podrán al día en esta materia.

Gracias a este manual sabrá cómo ampliar las aplicaciones de una base de datos con ODBC, cómo distribuir la base de datos en la web usando el conector de Base de Datos de Internet (IDC). Cómo entender temas de largo alcance como las transacciones, el procesamiento de éstas, la replicación, el almacenamiento de datos y muchos otros concptos que irá descubriendo a lo largo del libro.



Editorial: Prentice Hall
Nº páginas: 688
Nivel: Avanzado

Autor: Christopher S. Lester
Idioma: Español
Precio: 7.400 Ptas.(I.V.A. inc.)



Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

Instalé Linux el pasado mes desde la distribución RedHat que nos proporcionasteis con la revista. Después de algunos quebraderos de cabeza ya consigo arrancar las X-Windows correctamente con mi monitor y pantalla, y podría decir que estoy avanzando cada día más en mis andares con este Sistema Operativo del que todo el mundo hablaba.

La razón por la que les escribo es para hacerles una pregunta muy sencilla que todavía nadie de los que me aconsejaron instalar este Sistema Operativo me la ha justificado correctamente, aunque eso sí, 'ninguno duda de que sí sea compatible'.

La cuestión que les planteo es, ¿ Linux es compatible con el año 2000?. Supongo que será un asunto de Kernel más que de la distribución ¿no?. Muchas gracias y a ver si mis amigos están atentos a la respuesta.

Respuesta

En primer lugar comentar que nos alegramos de que empiece ya a disfrutar de Linux. Una de las razones por las que decidimos incluir la distribución RedHat 5.0 era por la sencillez del procedimiento de instalación, de manera que todos los profanos de Linux o aquellos que en su momento tuvieron problemas con distribuciones más complejas fueran capaces de introducirlo en su máquina sin ningún percance.

Sobre la pregunta que nos haces, bien, la respuesta es rotundamente sí. Cierto es que todos los que empiezan a dominar Linux le cogen un cariño especial y lo defienden con capa y espada, en este por lo menos tus amigos no andan nada equivocados.

Como la mayoría de los sistemas operativos Unix, Linux actualmente está escrito con representación de datos de 32 Bits. Esto nos sugiere que las dificultades de Linux empezarán a partir del año 2038 a menos que el Kernel, así como el código fuente asociado, sean migrados a implementaciones de 64 Bit capaces de manejar datos por otros 2 billones de años (creemos que es suficiente). Dado que los desarrolladores Linux tienen 40 años para corregir el problema, es muy probable que se implementen las soluciones bastante antes del año 2038.

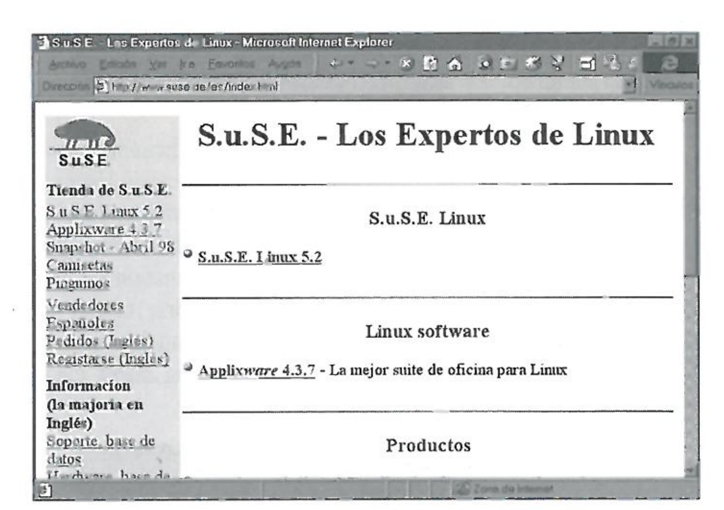
Por último, apuntamos que todo esto es un problema del Kernel, así pues en cuanto se corrija este para ser compatible con el año 2039, todas las distribuciones que se actualicen a este Kernel lo harán.

Si precisa más información al respecto (no es el primero que nos hace esta pregunta, ya que su duda se ha convertido en una de las más habituales), puede consultar las siguientes direcciones que corresponden a conocidas distribuidoras de Linux y que esperamos le ayuden en vuestra duda:

- Linux and Year 2000 (Christopher Browne) -> http://www.ntlug.org/~cbbrowne/linux10.html
- GNU Software in the Year 2000 -> http://www.gnu.org/software/year2000.html
- List of GNU Software and 1999/2000 Problems -> http://www.th.phys.rug.nl/~schut/gnulist.html
- Year 2000 Compliance: Lawyers, Liars, and Perl -> http://language.perl.com/news/y2k.html
- Open Group Desktop Technologies in the Year 2000 -> http://www.camb.opengroup.org/tech/desktop/faq/y2k.htm
- Apache Year 2000 Statement -> http://www.apache.org/docs/misc/FAQ.html#year2000
- Apache and the Year 2000 ->http://www.apacheweek.com/features/y2k/
- Caldera Year 2000 Project ->http://www.caldera.com/products/year2000/year2000.html
- Redhat Year 2000 Statement -> http://www.redhat.com/redhat/website.html#y2k
- Debian and the Millennium Bug -> http://www.debian.org/news.html#19980104
- Year 2000 FAQ -> http://ourworld.compuserve.com/homepages/rsandler/Y2KFAQ.htm

Pregunta

Hola a todos. Os felicito por la revista y en especial a Jorge Delgado, autor del estupendo artículo de sistemas distribuidos. No sé si es aquí donde respondeis dudas de todo tipo, pues la mía es extremadamente sencilla: ¿existen compañías de ordenadores que vendan Linux pre-instalado en la máquina?. Es decir, puedo yo ir a una empresa y decirles que no quiero ni Windows 95 ni NT, sino Linux y que el PC ya venga listo para funcionar con él.



La empresa alemana S.u.S.E. vende la pre-instalación de Linux.

Respuesta

Una pregunta curiosa de la cual te aseguramos que aquí en la redacción no teníamos una respuesta segura. Nos hemos informado al respecto y hemos averiguado que sí, aunque no en España. Hay una lista entera de distribuidores de ordenadores que permiten elegir como sistema operativo pre-instalado a Linux. La elección va desde un PC modesto basado en Intel hasta la gama alta como son los sistemas basados en Alpha o Sparc. Algunas compañías se especializan en una distribución particular de Linux mientras otras configuran cada sistema a las especificaciones solicitadas por el comprador.

Existen también otras compañías que se especializan en los llamados "cubos", que son sistemas pre-instalados y pre-configurados que no sólo incluyen Linux, sino además un servidor web, un servidor de correo, y todo el software y utilidades necesarias para poner en marcha un servidor Internet en unos minutos.

Por haber, hay incluso discos duros con Linux pre-instalado que son la solución perfecta para aquellos usuarios que están actualizando su hardware y quieren evitar otra pelea con la instalación de Linux.

Lamentablemente, como te hemos adelantado en estas líneas, no existe en España ninguna compañía que se dedique a esto, teniendo que recurrir a empresas como la conocidad S.u.S.E en Alemania (la más cercana), que vende ordenadores personales y notebooks con su propia distribución, S.u.S.E. , preinstalada en sus máquinas.

Existe una lista de distribuidores de ordenadores que permiten elegir como sistema operativo pre-instalado a Linux.

Pregunta

Un saludo, me llamo Alejandro Duque y tengo una duda acerca de los entornos gráficos de Linux. Por defecto, mi distribución Slackware me instaló el FVWM95, y aunque funciona bien, tengo aquí la última versión del KDE. La he visto en acción en unos JPGs y me parece mucho más vistosa y agradable, así que me he decidido a instalarla. Mi pregunta es sobre los pasos que tengo que seguir para activarla en el sistema, pues creo que tiene un poco miga el asunto.

Respuesta

Bien. La instalación de cualquier gestor de ventanas para Linux es bastante sencillo, aunque reconocemos que al principio puede ser un poco frustante y lioso.

Lo primero que debemos hacer es compilar las fuentes del mismo, como nos dices que el paquete que deseas instalar es el KDE, pues suponemos que has obtenido el conjunto ya pre-compilado; en el caso de que esto no fuera así, hay que acudir al fichero readme.txt en el cual encontrarás indicaciones sobre cómo realizar la compilación. En la gran mayoría de los paquetes también se suele adjuntar un script que realiza el proceso de manera desatendida.

Una vez tengamos los ejecutables, sitúalos en un directorio aparte, puedes

querer hacerlo en el /HOME/GUI/KDE por ejemplo, aunque debes usar un directorio diferente para cada entorno obligatoriamente.

Segundo paso a dar: vamos a crear un script personalizado para nuestro entorno, el cual será llamado a su vez por startx en lugar del fichero por defecto situado en /etc/X11/xinit/xinitrc que nos cargaría el FVWM95. El fichero se localizará dentro del directorio /HOME y tendrá por nombre .xinitrc con las siguientes líneas:

```
#!/bin/sh
xterm &
exec [ruta completa del gestor de ventanas]
```

A partir de este mismo instante, el script startx se encargará de encontrar este fichero y hará uso de él, arrancando las X-Windows con el nuevo entorno que le hallamos especificado en la ruta.

Pregunta

En estos momentos me encuentro programando una pequeña aplicación Unix en red y tengo que implementar unas funciones de búsqueda sobre archivos de texto de la base de datos. La cuestión es que creo que esto mismo lo podría realizar en un script, les agradecería muchísimo que me informaran acerca de la mejor elección posible.

Respuesta

Para realizar búsquedas de cadenas de texto en uno o más archivos ASCII, puede hacer uso de la conocida orden de Unix grep. Ésta es utilizada con dos mandatos, uno para especificar la cadena de texto a buscar y otro que indica el archivo o archivos en los que hay que realizar la búsqueda. Por ejemplo, grep hola datos nos devolvería por pantalla todas las líneas del archivo datos que contienen la palabra hola.

En contra de lo que pudiera parece a primera vista, la orden grep es muy potente y por supuesto admite todo uso de comodines o wildcards. En el caso de que quisiéramos realizar una búsqueda haciendo uso de ellos, deberá introducir el operando entre comillas.

Grep posee opciones tan versátiles como -l, la cual se encarga de imprimir una lista con todos los archivos en los cuales se ha localizado la cadena de texto introducida como primer operando o -i, que anula la sensibilidad a mayúsculas o minúsculas.

Si lo que de verdad te interesa es obtener más información a este respecto, te recomendamos que consultes los archivos man sobre el tema <man grep>.



Las herramientas más novedosas para programar



PROGRAMACIÓN

DirectX v5.2b

Último conjunto de drivers de Microsoft que implementan nuevas y mejoradas capacidades multimedia para su uso por vídeojuegos de última generación. En esta nueva revisión, se ha aumentado la conexión existente con Internet Explorer v4.01, de manera que páginas HTML visualizadas con este navegador pueden tener acceso a los mismos de una forma más directa.

Para aquellos que posean versiones más recientes de los drivers de vídeo o sonido de su tarjeta les recomendamos que instalen el paquete sin los mismos (tercera opción).

SDKs de Programación Internet

Conjunto de herramientas y SDKs de Microsoft para la programación de aplicaciones Internet usando las nuevas tecnologías creadas por Microsoft. Aquí podrá encontrar toda clase de programas y documentación adicional que le facilitarán el desarrollo de aplicaciones Internet de cualquier tipo; desde cómo realizar el acceso a las bases de datos más famosas a cómo incluir soporte de videoconferencia a través de Netmeeting.

PowerBuilder Code Analyzer v1.2b

Herramienta que rastrea el código fuente de aplicaciones PowerBuilder para verificar la compatibilidad con estándares corporativos, de programación y del año 2000. Proporciona un análisis detallado con los resultados, de manera que es más sencillo modificar el código para que cumpla los citados estándares.

PowerBuilder Code Analyzer contiene un localizador de variables avanzado que puede usar para situar fácilmente todas las variables (o simplemente los tipos de datos seleccionados) en los *scripts* y ventanas de datos de las aplicaciones.

Catalyst 3 Application Generator

Utilidad de desarrollo diseñada para generar rápidamente aplicaciones de procesamiento de datos portables. Está dirigida para entornos de tamaño medio que usen tecnología de bases de datos cliente-servidor, estaciones de trabajo inteligentes, redes y múltiples sistemas operativos.

Catalyst 3 genera aplicaciones de procesamiento de datos portables

Catalyst en un entorno centralizado de datos. Uno de los objetivos del Lenguaje Catalyst es proteger al desarrollador de las complejidades de los distintos interfaces gráficos de usuario, interfaces de datos del servidor y de los lenguajes de programación de bajo nivel. Con Catalyst, el desarrollador posee una utilidad en la cual él mismo puede especificar el procesamiento de los datos en un nivel conceptual alto, acelerando el proceso de desarrollo significativamente y haciendo que los sistemas destino sean menos dependientes de las plataformas de operación.

Genitor v3.2 System Installation Kit

Tanto si acaba de empezar a programar en C/C++ o es ya un experto programador, Genitor puede ayudarle a alcanzar nuevas cotas en el desarrollo C/C++, y además incrementa su productividad permitiéndole crear un producto de más alta calidad.

Genitor es una suite de programas que ayudan en la construcción, documentación y mantenimiento de código C/C++. Su

potencial se extrae al máximo en grupos de trabajo pero también puede ser usado por programadores individuales.

Posee un entorno gráfico de edición que permite la rápida construcción de clases C/C++, funciones, estructuras, uniones y plantillas. Este entorno le permite definir objetos mientras piensa a un nivel más alto. Usando Genitor se puede concentrar más en conceptos que en la sintáxis del código, ya que éste se encarga de generar código listo para compilar y en varios formatos posibles.

La versión que aquí incluimos precisa de un código llave para su evaluación. Se puede obtener el mismo en la dirección: **http://www.genitor.com**

REDES

OpSession 2

OpSession es la solución definitiva para proporcionar colaboración en tiempo real y soporte de software a sus usuarios dentro de grandes corporaciones. OpSession permite compartir directamente pantallas de aplicaciones, ficheros, y datos específicos de cada sistema.

Steelhead permite compartir módem a través de una red

La herramienta funciona sobre redes TCP/IP y puede visualizarse desde la ventana de un navegador HTML. Permite a sus usuarios colaborar entre ellos independientemente de su localización física o su plataforma hardware.

La aplicación se compone de dos módulos, un servidor o *host* y de un cliente o visor. El servidor sólo se ejecuta bajo Windows 95 y NT 4.0, los clientes son de libre distribución y puede encontrarlos para casi todas las plataformas en el directorio \REDES\OPSES.

Protocolo PPTP para Windows 95

Actualización en inglés de Dial-Up Networking (Acceso Telefónico a Redes) de Windows 95 que activa el soporte para el reciente protocolo PPTP (Point-to-Point Tunneling Protocol) que permite utilizar Internet como su propia Red Privada Virtual (VPN: Virtual Private Network).

OpSession 2 puede visualizarse desde un navegador HTML

Con PPTP, sus usuarios pueden conectarse a través de su propio proveedor a Internet y acceder a su red de una manera sencilla y segura, como si estuvieran en los escritorios de su empresa.

Entre otras características importantes se encuentran:

- Escriptación robusta de datos y compatibilidad absoluta con protocolos de red como IP, SPX y NetBEUI.
- Soportado por Windows NT Workstation y Windows 95.

Antes de instalar el mismo, debera actualizar su versión de Winsock.

Steelhead

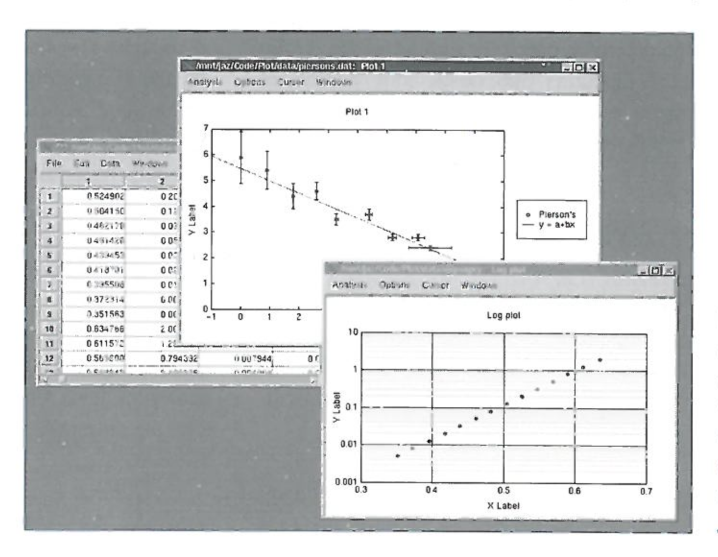
Steelhead es el sobrenombre de un servicio de NT que permite compartir un módem a través de una red de manera que se puedan realizar conexiones a Internet de forma eficiente y bajo demanda.

Al instarlo, se añadirá un nuevo dispositivo de conexión a la red que será un módem o un adaptador y nos permitirá definir el tipo de enrutamiento que se realizará con él. En la configuración del sistema habrá que definir los ordenadores clientes para que utilicen dicho adaptador como *Gateway*.

FTPMax v3.0

FTPMax es un servidor FTP 32 Bits multitarea para Windows NT y Windows 95. Posee características muy llamativas en lo que a administración se refiere pues permite el mantenimiento de ficheros mediante arrastrar y soltar así como soporte multi-dominio, usando árboles virtuales para cada uno.

Mediante esta aplicación, la asignación de espacio y acceso a directorios y ficheros se hace más sencilla que nunca. Permite asignar un mensaje diferente de saludo y despedida para cada dominio, así



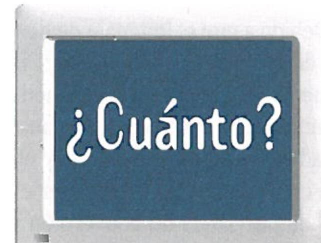
Genitor es una suite de programas que ayudan en la construcción, documentación y mantenimiento de código C/C++.



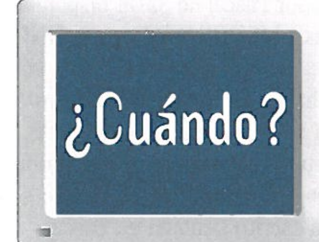
INTERNET ASEQUIBLE

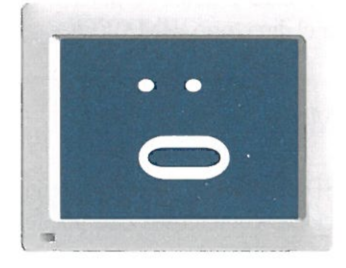


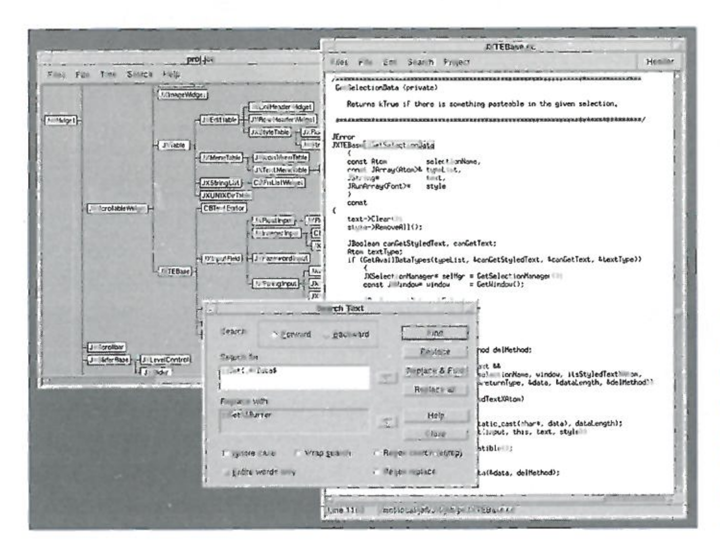












JX ha sido construido directamente por encima de Xlib y optimizado para obtener un excelente rendimiento.

como adjuntar otro adicional a ciertos usuarios.

Los registros de FTPMax son guardados usando ODBC con Microsoft Access o SQL Server. Puede usar su base de datos ODBC preferida para crear informes adicionales personalizados.

Además, incorpora una opción denominada *Snoop Mode* que nos proporciona información interactiva sobre lo que está haciendo cada usuario en tiempo real.

UNIX

ProvideX v4.01

ProvideX es un potente sistema de desarrollo de aplicaciones que ofrece al programador la capacidad para transformar sencilla y rápidamente su código existente en una suite de aplicaciones que le situará a años luz de la competencia, tanto en los aspectos visuales como de funcionalidad.

ProvideX incluye una gran variedad de modos de emulación

Debido a que el lenguaje de desarrollo se deriva del Business Basic, ProvideX puede ejecutar aplicaciones con cambios mínimos. ProvideX incluye una gran variedad de modos de emulación y utilidades de conversión para simplificar el proceso. Las utilidades de conversión están disponibles para varios sistemas Business Basic incluyendo BBx, Open Basic, Thoroughbred, etc.

Se pueden encontrar otras versiones para MSDOS, Windows y SCO Unix además de Linux en el directorio \UNIX\PROVIDEX.

Free Pascal v0.99.5

Free Pascal (también FPK Pascal) es un compilador Pascal de 32Bit disponible para Linux y MSDOS bajo licencia GNU modificada. Esto quiere decir que sólo se puede hacer uso de las librerías estáticas cuando se crean aplicaciones DOS, aunque es posible disponer del código fuente del compilador, que curiosamente se encuentra escrito también en Pascal.

El código fuente así como las versiones para DOS y Linux se pueden encontrar en el directorio \UNIX\FPK.

MpegTV SDK v1.0

Este SDK permite desarrollar fácilmente aplicaciones comerciales que pueden hacer uso de las librerías de MpegTV para reproducir vídeo MPEG-1 con sincronización de audio en tiempo real.

El complejo código necesario para reproducir MPEG en tiempo real es parte de las librerías y para sacar partido del mismo basta con usar las sencillas APIs que están disponibles en casi todos los sistemas UNIX.

MpegTV SDK v1.0 está disponible actualmente para:

- Linux x86
- Linux Alpha (nuevo)
- Solaris SPARC
- Solaris x86
- HP-UX
- BSD/OS 3.0

y se puede encontrar en el directorio \UNIX\MPEGTV10.

Free Pascal es un compilador de 32 Bits disponible para Linux

Para acceder al período de evaluación del producto por 30 días es necesario solicitar una licencia personal en la dirección www.mpegtv.com

JX

JX es un entorno completo de desarrollo C++ para el sistema X-Windows que proporciona soporte para todas las facetas de un proceso de desarrollo, incluyendo aplicaciones distribuidas. Ha sido construido directamente por encima de Xlib y optimizado para obtener un excelente rendimiento.

JX viene con una suite completa de pruebas que muestran todas las características de la librería y un conjunto de programas ejemplo que introducen los conceptos avanzados. Esto proporciona una fuente muy rica de código ejemplo, tanto

a la hora de desarrollar aplicaciones de alto nivel como para la programación de clases individuales.

Se soportan los siguientes sistemas operativos:

- RedHat 4.2, 5.0 ó 5.1
- SunOS 4.1.4

Todas estas versiones de pueden encontrar en el directorio \UNIX\JX.

VARIOS

Network Undelete v1.0

Utilidad que recupera instantáneamente ficheros borrados accidentalmente. Se integra como la Papelera de Reciclaje de Windows pero amplia su funcionalidad a toda la red.

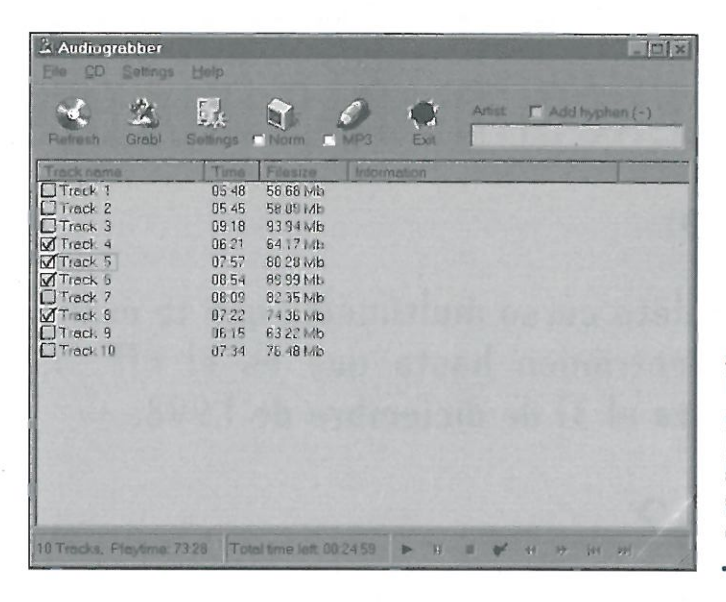
NU soporta privilegios y permisos NTFS

Se trata de la única utilidad del mercado que puede restaurar ficheros sobre una red. Captura todos los ficheros borrados tanto desde el Explorador de Windows como desde la línea de comandos.

Network Undelete soporta privilegios y permisos NTFS.

Diskeeper v3.0 & Diskeeper Lite

Diskeeper es un defragmentador de archivos para Windows NT que soporta NTFS. Entre sus características principales merece la pena que destaquemos ente otras, sus controles de red y su funcionamiento en modo desatendido pudiéndose lanzarlo periódicamente a una hora y fecha concreta.



Audiograbber copia los datos directamente del CD lo cual permite hacer copias perfectas de los originales.

En la actualidad es el único desfragmentador del mercado para red. Además de las versiones de evaluación, suministramos una versión reducida del mismo para una sola máquina cuyo nombre es Diskeeper Lite.

También, puede encontrar otras versiones de Diskeeper Lite para Windows NT sobre otras plataformas en el directorio \REDES\DISKEEP\LITE.

Luckman's Anonymous Cookie Beta2

Luckman's Anonymous Cookie es un programa de libre distribución que permite desactivar instantáneamente todas las cookies de su navegador HTML.

Este modo anónimo de funcionamiento asegura una total privacidad puesto que esconde toda la información personal a los servidores web remotos. Podemos activarlas de nuevo en cualquier momento de nuestra sesión Internet, simplemente debemos de pulsar con el ratón en la opción adecuada (sólo para Internet Explorer).

Audiograbber v1.20

Audiograbber es una elegante aplicación que graba las pistas de audio digitales de un CD. Copia los datos directamente del CD (y no a través de la tarjeta de sonido) lo cual permite hacer copias perfectas de los originales. Posee opciones para realizar tests de manera que se pueda comprobar que la copia es exacta. También permite normalizar el volumen de las copias, borrar silencios del principio o final de las pistas y enviarlas a una aplicación externa (o utilizar el codec interno) para su posterior compresión a MP3.

Diskeeper es el único desfragmentador del mercado para red

Audiograbber funciona bajo Windows95 / 98 y Windows NT. Soporta unidades de CDROM ya sean IDE o SCSI (aunque hay algunas excepciones en estas últimas). Ha sido probado con unidades de marcas como NEC, Pioneer, Plextor, Sony, TEAC, Toshiba y Yamaha. Aunque puede funcionar con otras de otros fabricantes.

NOTA: Los usuarios de Windows 95 OSR2 deberán copiar al directorio \SYSTEM\IOSUBSYS de Windows 95 el driver scsi1hlp.vxd para usar DAE (Direct Audio Extraction) con Audiograbber. Recomendamos hacer una copia de seguridad antes de reemplazarlo. Adjuntamos así mismo una utilidad para eliminar los clicks de los .WAV.